ZOO

SOA

SOA

项目综述

衷心感谢 IBM 举办的这次大赛，竞赛题目和我们的研究兴趣如此契合，以至于在开赛之初我们就认识到，这不但是我们千载难逢的契机，而且是一个加强学术界和产业界合作，联手推进 SOA 成熟发展的机会。我们知道，目前关于 SOA 的发展存在两种观点：工业界认为 SOA 到现阶段已经基本成熟，应该把注意力集中在 SOA 的实施和落地上，为此应该完善相关的方法论和配套工具，以及总结出一套最佳实践和最坏实践，应该说本次大赛就是这样一个目的；而学术界认为 SOA 远谈不上成熟，还有很大的进步空间，比如说 Semantic Web 技术和 Web Services 技术结合，催生出了 Semantic Web Services 的全新领域，WSMO，OWL-S，METEOR-S 等项目更是承诺要使 SOA 更加自动化和智能化。

在这样的背景下，我们来参加这次大赛，就是想尽自己微薄的力量，把学术界最新的思维和成果应用到工业实践中，看看能不能对 SOA 的发展有所贡献。在这之前，我们的研究组完成了一篇关于开放环境中服务发现的论文[1]，并计划基于 Georgia 大学和 IBM Watson 研究院的研究成果 METEOR-S 开发自己的服务发现、选择和组合平台（我们称之为服务集市）。所以当看到本次大赛中关于智能信息服务社区的扩展需求时，我们无法按捺不住心中的激动：这正是我们一直想做的事情！

我们的文档分成两条主线：以 107 页为界，之前的内容用当前最佳实践，以及经过验证的方法和技术分析设计凤凰公司案例中的 SOA 架构。包括业务建模的 CBM，EA；流程建模的 BPM；电子商务模式的采用； SOA 成熟度分析等；还有服务设计的 SOMA，SOAD 等分析建模方法；基于 ESB 的 SOA 参考架构等。这部分内容是规定动作，每支参赛队伍都会做的，而我们力求做细做实。

我们真正的突破点是在 107 页之后。在“SOA 语义化”一节，我们首先以实例分析了 SOA 不是银弹，它还面临许多挑战；接着指出语义化是解决这些挑战的最有效的方法。在后面的篇幅里，我们从不同的角度，不同的层次详细介绍了 SOA 语义化的方方面面。通过智能信息服务社区来展示我们进入这次 SOA 大赛中真正的特色思路。前面提到过参赛之前我们正打算做“服务集市”平台，在该部分内容中，我们介绍了平台的概念，设计和架构，并且节选了本研究小组学术论文的部分章节对主体服务模型的本体描述做了详细解释。

在设计实施计划里，我们给出了实现这个平台所用到的软件和工具。特别值得一提的是 METEOR-S，它是语义 Web 服务领域的一个轻量级解决方案；此外还有 Jena 和 Protégé，都是开发语义本体不可或缺的基础。

另外值得一题的是我们的项目管理和团队建设。详情请看“大赛项目管理文

[1] L. Liu, et al. Strategic Capability modeling of Services. Proceedings of the 2nd Service-Oriented Computing and Consequences to Requirements Workshop, at the Requirements Engineering Conference, 2006. To appear.

档”一节和“后记”。在后记中，我们总结了参赛 3 个月来的点点滴滴，这些真情流露的文字，是我们此刻最真实情感的表达；不管岁月如何流逝，这次经历将随着这些文字，永远沉淀在我们的心里。

目录[2]

[2] 为了方便您的阅读，我们把相对重要的地方用粗体标出。

业务模型分析设计

1. 凤凰公司现状分析及面临的挑战

自从 2000 年成立以来，凤凰公司经历了爆炸式的成长，从一个医疗设备制造领域默默无闻的小公司，一举成为行业中的领导者。这些成就可以归功于公司那强大并无处不在的销售渠道，和所有员工对品质的孜孜追求。然而公司高层知道，这些成就很大程度上得益于对企业信息化建设的重视。 2004 年凤凰公司引进并在公司内部成功实施了 ERP 系统，主要用于财务管理，产品库存及订单管理等。ERP 的实施大幅度地提高了公司的管理效率，实现了零库存。 2005 年 8 月份凤凰公司又引进并成功应用了在线客户关系管理系统，此举更是大大提高了销售人员的工作效率。正是因为这些信息系统的支撑，使凤凰公司能够以比竞争对手低得多的成本，给客户提供高质量的产品和服务。

进入 2006 年后，CIO 张总发现情况有些不对劲。由于 ERP 和 CRM 使用的是不同厂商的系统，它们分别维护自己的产品和客户信息数据，使销售人员和生产管理人员不能以统一的视角平滑合作。

更为头疼的是并购带来的问题。今年年初，公司为了进入心脏起搏器市场，斥巨资收购了该行业的领导者青龙公司。就在整个公司都为顺利进入这个全新的市场欢呼时，张总发现事情并不像想象中那么简单。青龙公司有一套自己的 ERP 和 CRM 系统，除此之外，他们还有自己的资产管理和人力资源管理系统。在可预见的将来，肯定还要并购更多的公司，每次并购都存在同样的问题。怎样把这些应用整合进凤凰公司原有的系统，成了张总最担心的一件事。

万合公司的专家给张总画了一张图，用来描述凤凰公司现有的状况:

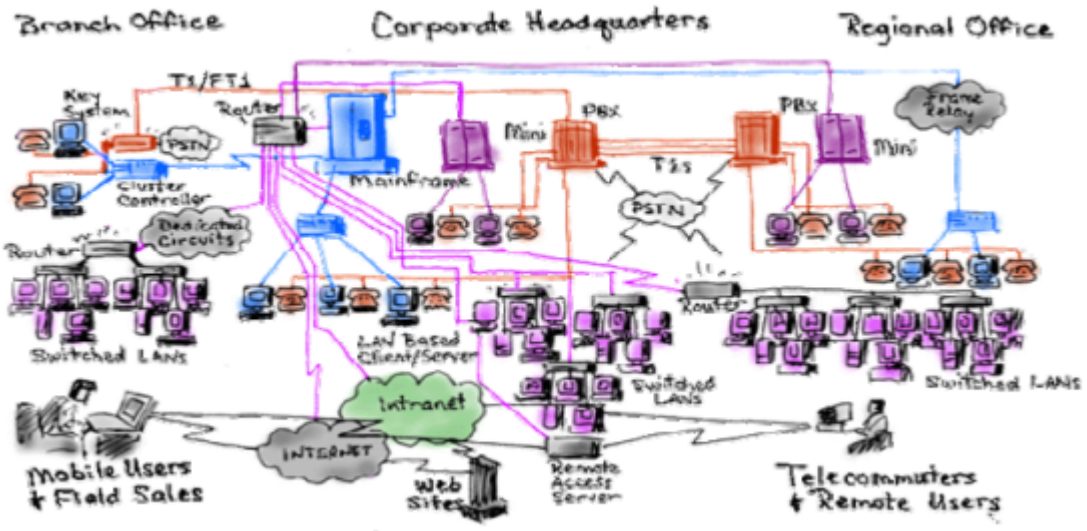


图 1：凤凰公司信息系统示意图

source:IBM Busniess Consulting Services

在万合专家的眼中看来，凤凰公司的扩张过于迅速，并且没有一个统一的规划，通常是需要什么系统就上什么系统，既没有考虑到怎样和以前的遗留系统整合，也没有考虑到以后的扩充，这就造成了现在这种一团乱麻的局面。在企业规模还不是很大的时候，这还不是大问题；但随着企业的成长，这种缺乏灵活性的架构会带来致命的问题。比如管理越来越多的异构的，独立的遗留系统；整合重复的业务功能；统一不同的服务标准等等。总之，凤凰公司需要一个灵活的 IT 基础架构，来支撑飞速变化的业务需求（On Demand）。

基于这些考虑，专家给张总提出了面向服务的构架（SOA）。 专家们特别指出，要成功实施 SOA，必须遵循一套严格地实施计划。首先建立起公司的业务组件模型。万合公司有一套行之有效的业务组件建模方法 CBM（Component Business Modeling ），曾经在很多案例中起到非常重要的作用。基于万合公司的良好声誉，张总同意用该方法对凤凰公司的业务组件进行标准化。在此基础上，对选出的热点组件用 EA 的方法进行分析，从不同的视角对业务建模。最后分析和现状的差距，制定出转化计划。需要注意的是，这一系列步骤都必须在严格的管控下施行。如图 2 所示。

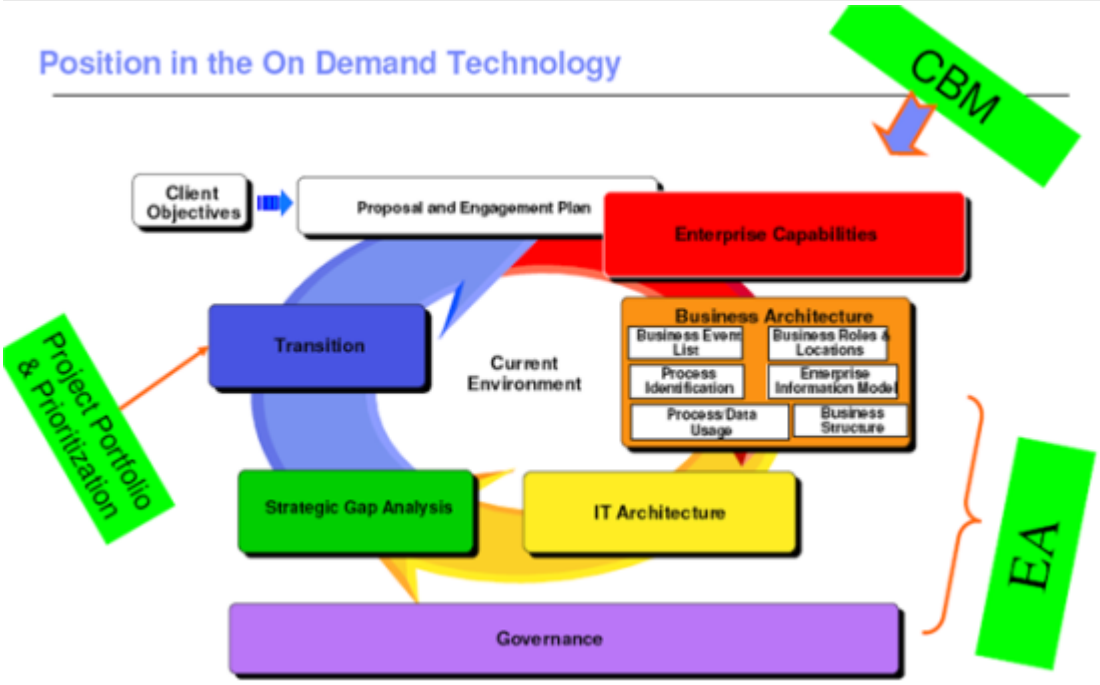


图 2: 向 SOA 转化的路线图

source:IBM Busniess Consulting Services

2. 业务组件建模

2.1 CBM 介绍

目前，很多企业都意识到了必须优化自己的业务，但是缺少一个有效的分析方法。传统上比较常用的技术是企业流程再造（Business Process Reengineering），它用了一些线性规划的方法，可以大幅度优化工作流的效率。但是对于找出散布在各个流程中的相似的活动，它就有些无能为力了。

业务组件建模（CBM）给企业提供了一个寻找业务操作的简单方法。它把混杂在各个流程中的操作抽取出来，这样可以清楚地看出每个操作会对企业带来什么价值。然后把相似的操作组织成统一的，独立的模块，由这些模块组成整个企业。这种把业务活动看作自治组件的视角，非常有利于决策者打破历史形成的壁垒，这些壁垒以前在组织内不同的单位，渠道，产品线，甚至不同的地理位置间都是普遍存在的。

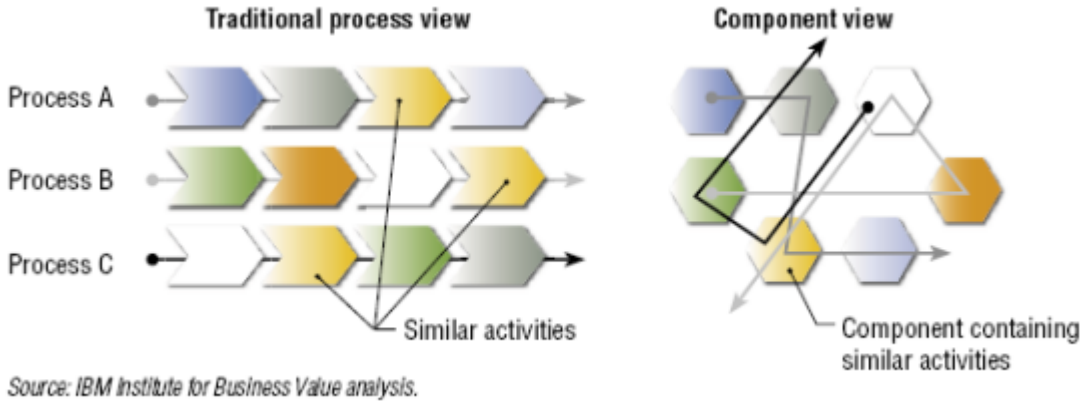


图 3：流程的视角和组件的视角

source:IBM Busniess Consulting Services

例如凤凰公司传统的 X 光透视机产品线有自己的订单处理系统，新并购的青龙公司的心脏起搏器产品线也有自己的订单处理系统。从流程的角度，这是两个完全不同的流程；但从功能的角度看，他们完成的功能是一样的。那么把这两个系统整合为一个单一的组件，可以显著降低企业的成本并提高效率。组件视角如图 3 所示。

2.2 业务组件定义

所谓业务组件，是指把一组紧密耦合的业务活动打包成的一个有明显边界的单元。除了相关的业务活动外，还包括业务活动所需要的人力，技术，数据等资源。组件最本质的属性是它所提供的服务，而不是它在流程中的位置。所以应该把业务组件看作整个企业价值链（网）中离散的节点。组件应该有定义良好的接口。每个组件都应该包括从其它组件接受输入，进行处理，输出结果到其它组件的功能。组件间标准的接口可以避免企业把业务功能混为一团乱麻。

业务组件模型可以帮助企业建立 On demand 的思维方式。把业务看成一个由不同组件组成的网而不是线性流程，可以帮助管理者发现哪些组件会带来价值，哪些不会。这样当企业必须立刻对外界环境的变化作出反应的时候，管理者可以马上找出那些带来最大价值的模块加以调整。

业务组件间的边界是由它们需要的服务和提供的服务分隔的。这种分隔可以使组件在集成在一起的同时，保持相对的独立性，甚至可以把该组件设置成一个子公司。

组件的例子如图 4 所示。这是一个制定市场细分计划的业务组件，负责市场分析和客户定位。它用到的服务包括业务规划和市场追踪，提供的服务包括产品组合升级和市场细分定位。

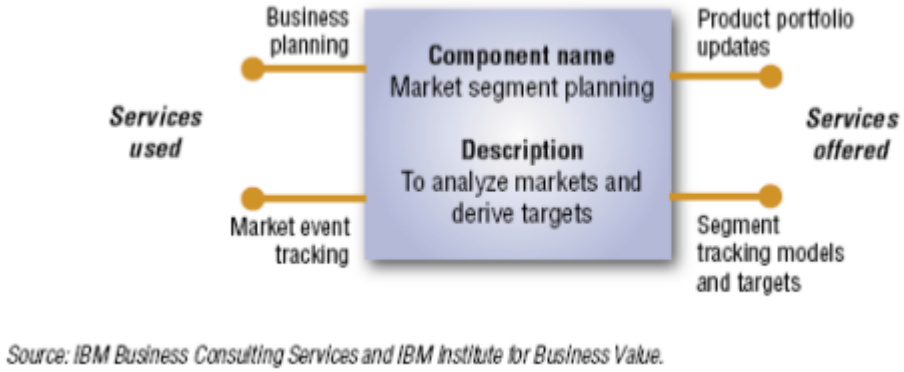


图 4: 业务组件示例

source:IBM Busniess Consulting Services

2.3 业务组件模型给凤凰公司带来的利益

为了维持企业竞争力，凤凰公司需要这样一种业务模型:

- 可以适应快速，持续的变化。
- 既要处理大规模的业务，又要足够灵活。
- 保持顾客长期的忠诚度。

为了达到这些经常互相冲突的目标，企业不能再把业务看成一个整体，而应该是功能的集合。这些功能为了完成共同的目标互相配合创造价值，而且随着外界的变化动态增减进行调整。

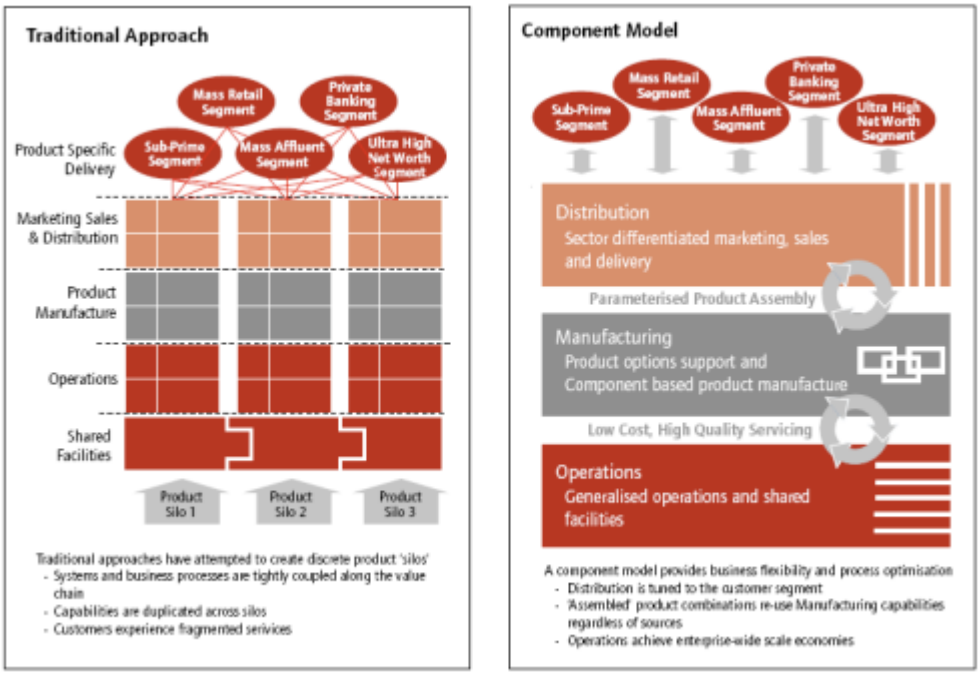


图 5：从竖井到组件

source:IBM Busniess Consulting Services

CBM 正好满足了这些需求。组件通常集成了相关业务活动，用到的信息系统，流程，组织结构，绩效度量等内容，是一个独立的实体。为了实现到 CBM 的转化，凤凰公司必须把价值链打碎成零散的组件，从而消除企业中的条块分割，即所谓的“竖井”结构，如图 5 所示。

需要注意的是，尽管每个组件都是高度自治的，但只要遵守共同的服务约定，就可以平滑的链接起来。所以 CBM 的实施，需要高度灵活的 IT 架构，由于 SOA 自身的特性，使它非常适合这种角色；反过来，SOA 的实施，也必须理清企业的功能划分，识别出合适的 Service，而这是 CBM 所擅长的。所以，CBM 和 SOA 互相配合，彼此促进，是密不可分的。

2.4 凤凰公司业务组件模型

凤凰公司的业务组件图如图 6 所示。横轴代表业务功能，分为业务管理，新业务开发，关系管理，服务销售，产品制造和财务控制六大块。纵轴表示操作级别，最高层负责战略导向，第二层负责执行控制，最后一层负责具体执行。横纵轴相交的每个节点都代表一个业务组件。比如服务销售的业务，最高层的组件负责销售规划，控制层的组件进行销售管理，执行层分为 3 个组件，分别为销售，客户交涉和客户联系。

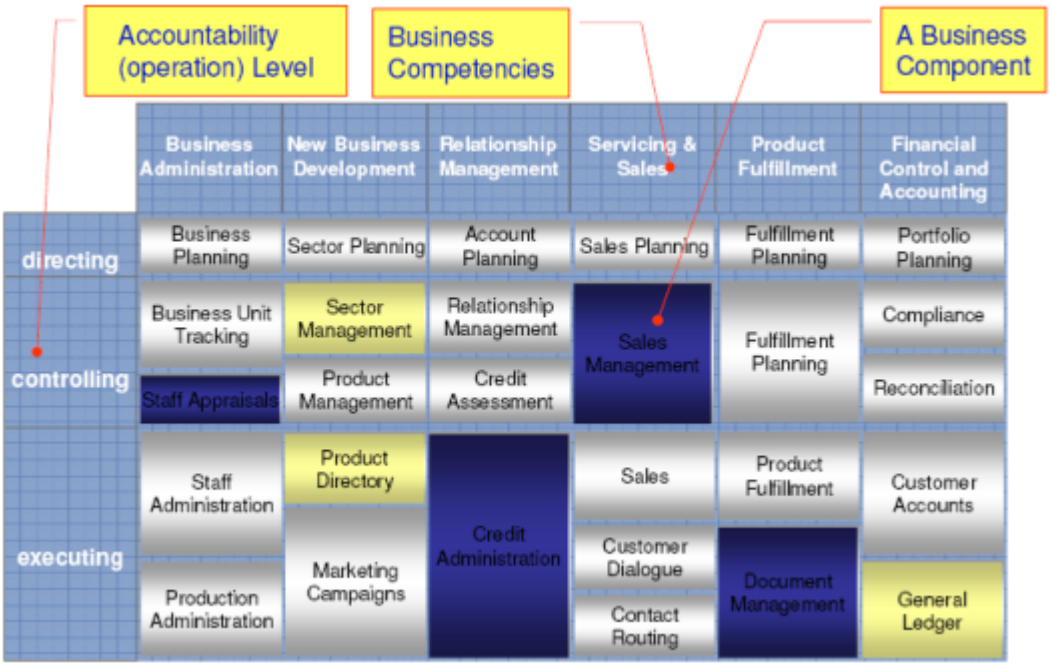


图 6: 业务组件图

source:IBM Busniess Consulting Services

根据这张图，万合公司帮凤凰公司把所有的业务分成这 28 个组件。为了使业务转化更有目的性，万合的专家们请张总标定出几个最紧迫，最有价值的组件首先实施。张总根据每个组件的成本和它们带来的价值，认为目前最应该改善的是销售人员的效率，所以销售组件被标定为最“热”的组件首先进行分析。

2.5 热点组件分析

Component	Activity Level 1	Strategic Impact	Financial Impact	Transformation potential
Sales	寻找商机 商务谈判 报价 填写订单 签合同 收款	Differentiating	High cost	High potential
Sales Management	市场调研 审批报价单 审批合同 制定销售计划 制作销售报表	Basic	Limited	Medium potential
Consumer Rela	管理呼叫中心	Differentiating	High cost	High potenti

tionship Management	销售/递送状态跟踪 客户关系信息维护 客户商业价值评价 信用管理 客户偏好挖掘			al
Financial Management	产品成本控制 税率管理 订单处理 票据管理	Competitive parity	Limited	High potential
Warehouse Management	库存规划 存量检查 出入库管理	Competitive parity	High cost	High potential
Delivery Management	物流渠道管理 发货管理	Basic	High cost	Medium potential

图 7: 热点组件分析

根据图 7，纵列是挑选出的几个热点组件，横列分别是它们所包含的 level 1 的活动，以及对它们的战略影响，财务影响和转换潜力的评价。可以明显看出，Sales 组件尽管成本很高，但有很大的战略价值和转换潜力，应该优先实施。类似的，应该优先实施的还包括财务管理组件和库存管理组件。

有了要分析的热点组件后，就要找到一种合适的分析方法。万合公司的专家们建议用企业架构（Enterprise Architecture）的方法，它可以从多个不同的视角看待企业，用一种统一的视角分析企业。

3. 企业架构分析

3.1 企业架构简介

有两句俗语说的很好，第一句是如果不知道自己在哪，地图没有一点用；第二句是如果不知道自己去哪，任何路也不会有。对于企业也一样。企业架构（EA）就是为了解决这两个问题：告诉企业在地图中的位置；告诉企业前进的方向和如何前进。可以将 EA 比作城市规划，它要为所有的业务单元提供一个统一的视角；为业务系统的设计和实现提供一个一致的方法，并且有能力及时相应外界变化。

EA 通常包含以下内容：总体的实施规划和原则，用于战略指导；一个框架，用来表示架构中的基本组成模块；管控措施，用来保证企业架构正常工作；最后，可以采用参考架构，用一些成熟的架构作为蓝本来节省时间和提高质量。

自从1985年德国的Zachman教授首先提出EA的概念以来，经过20多年的发展，已经形成了几个成熟的框架，包括Zachman框架；开放组织的The Open Group

Architecture Framework；美国国防部的DoDAF；美国财政部的联邦企业架构框架（FEAF）等等。虽然它们在细节上有些不同，但本质上是类似的，它们都要保证在IT的支持下，从高层战略到具体业务方案的实现。

EA通常分为业务架构（Business Architecture）和IT 架构（IT Architecture），它们之间存在一条巨大的鸿沟，分析师的工作就是如何填平这条鸿沟。如图8所示，企业的最上层是战略层，包括业务战略和IT战略，分别负责寻找新的商业机会和合适的IT技术；中间规划层都属于EA的范畴，类似于城市规划。其中业务构架是从业务的角度建模，包括流程，信息，人力资源等等，而IT架构是从应用，数据，技术的方面考虑。怎样从业务平滑映射到IT，是目前面临的最大挑战。这张图的最底层就是如何把整体规划落实到一个个具体的项目，就像城市规划好之后设计具体的建筑。

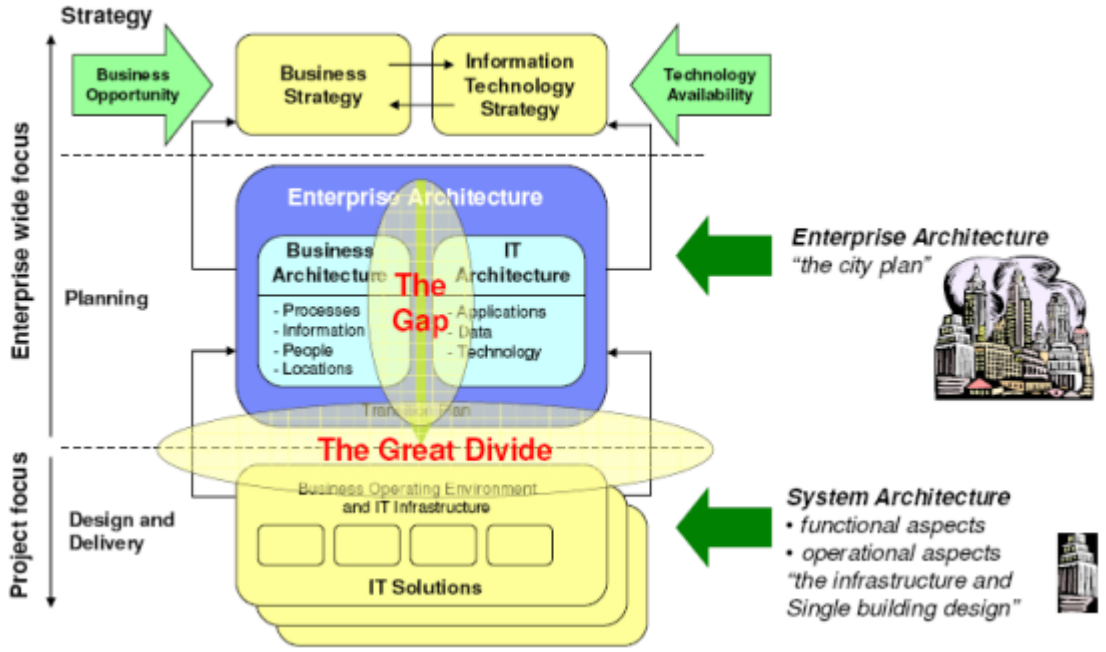


图 8：EA 在企业中的位置

source:IBM Busniess Consulting Services

3.2 业务构架分析

业务架构给企业的业务提供了一个统一的，稳定而又富有弹性的定义，它回答了业务运行中的一些根本性问题。由于企业中不同角色对同样问题的看法千差万别，不能一概而论，业务构架提出了从五个视角看待企业业务。它们包括：

业务活动　这个视角主要关心企业进行的能力，它试图回答这几个问题：业务活动是怎样定义的？它可以被容易的改变吗？它可以重用吗？

所谓业务活动，是指为完成某项工作而进行的一个手工或自动的操作。这些活动被组织成逻辑上的层次结构，从最顶层的活动可以一直分解到最小的逻辑工

作单元。除了自上而下的分解，在建立活动模型的过程中，也可以混合使用自下而上的技术，从最底层的工作单元逐层组合。

业务信息 既然信息是企业中最重要的资产，那这些信息是以一种统一的，可用的形式定义的，还是杂乱冗余未定义的？它能被有效地使用吗？只有特定的组织结构内的人员才能访问它吗？这些都是业务信息模型试图解答的问题。

通常企业的信息模型都会用 ER 图来表示。信息被组织成不同的抽象层次：最抽象的一层是主题域，用它把一组相关的实体组织起来，以便管理和实现；其次就是实体和实体之间的关系，一切需要保存成数据的事物都可以看作是实体；最后是属性，一切业务关心的实体的某个方面都是属性。

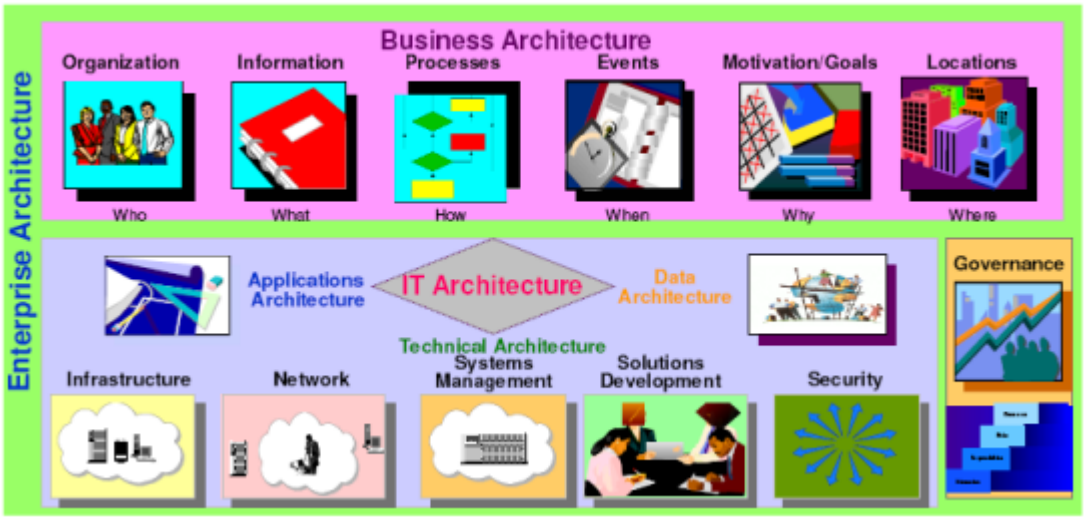


图 9：EA 的组织结构？

source:IBM Busniess Consulting Services

3.3 凤凰公司业务活动模型

业务活动就是一个自动的或者人工的完成某项单元工作的操作。逻辑上的业务活动一般按照逻辑结构组织起来。对于功能活动来说，可以将它们逐步分解直到找出最小的工作单元。一般来说，业务活动建模的主要的手段是功能分解，但是也可以采取自下而上的方法。行动本身可以被组合起来构成业务过程，因为业务过程本身可以看成另一种形式的活动。

对于企业来说，业务活动建模的关键在于找出企业的基本功能单元，这些功能单元按照某种共性进行分组，但同时又可以跨组进行合作，进而组成业务流程。业务活动的关键在于找出基本的功能。

下面是凤凰公司完成日常运转需要的基本业务活动建模:

功能：销售

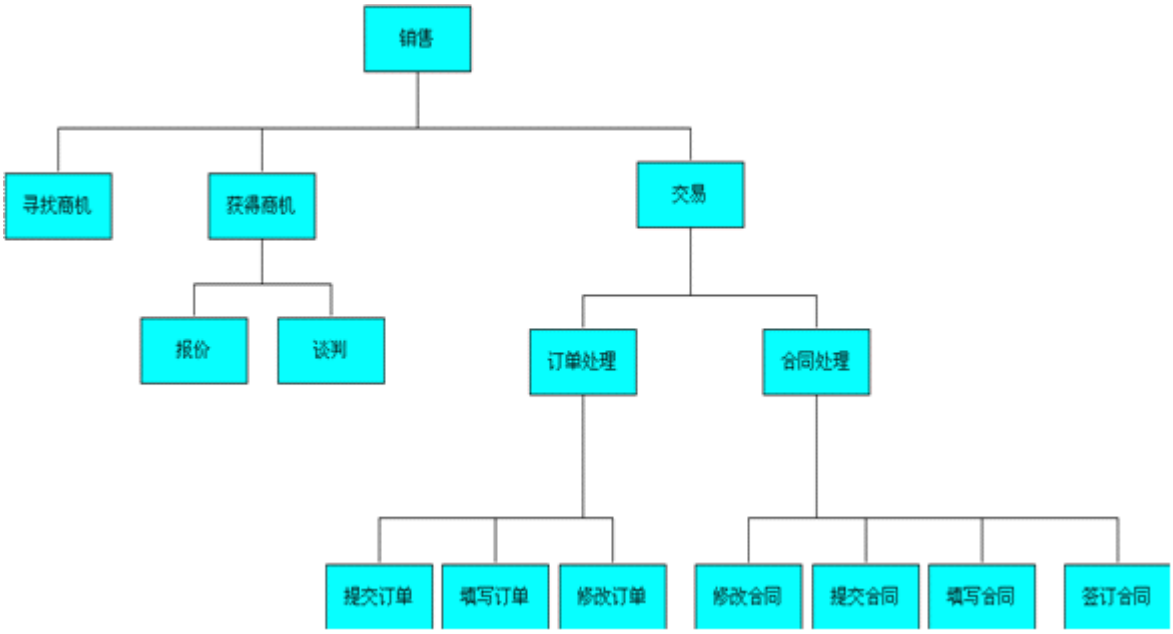


图 10 销售功能分解图

功能：销售管理

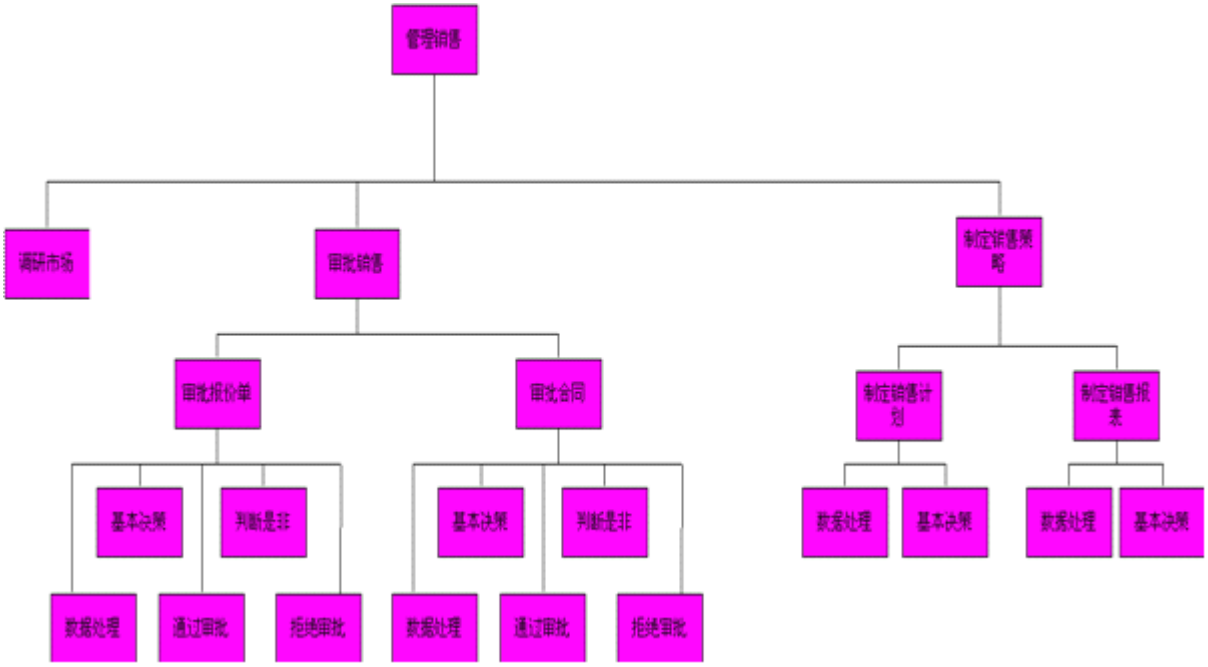


图 11 销售管理功能分解图

功能：客户管理

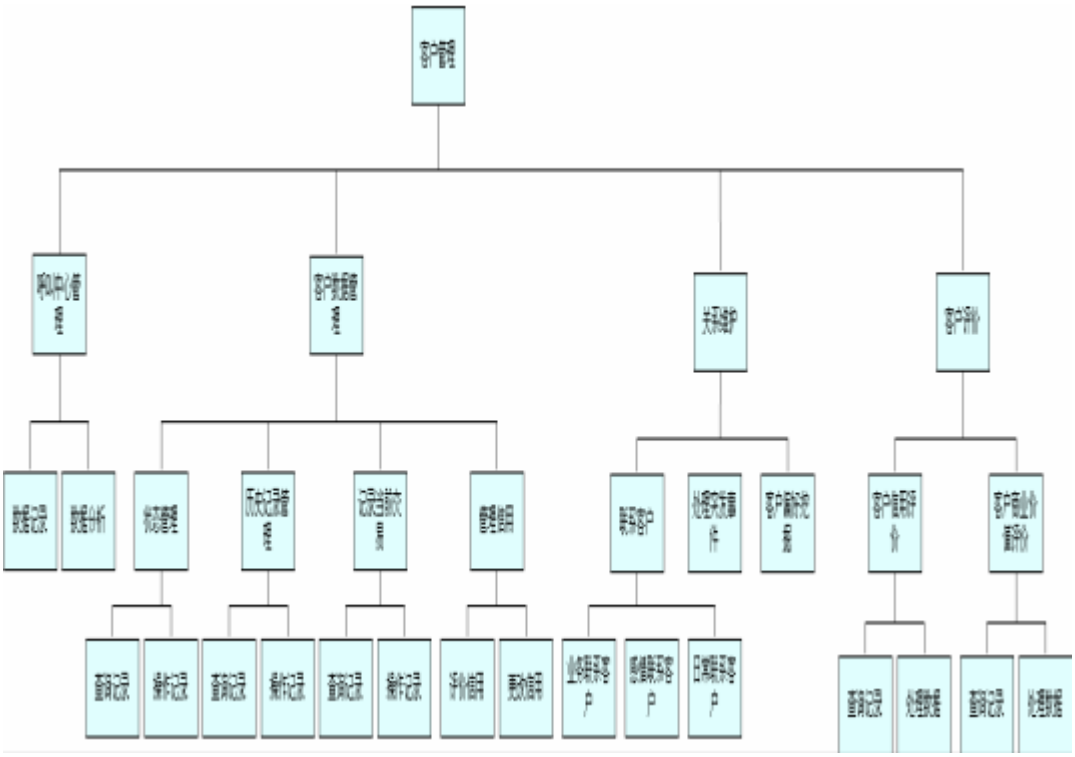


图 12 客户管理功能分解

功能:库存管理

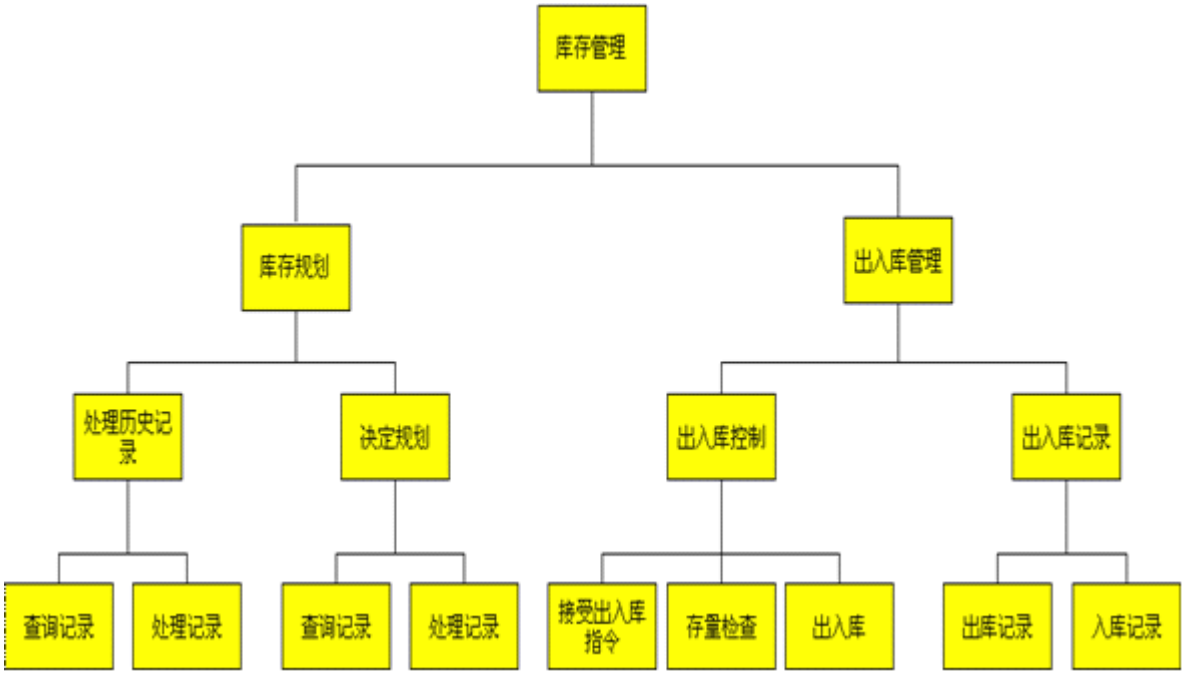


图 13 库存管理功能分解

最底层的活动是粒度最小的活动，可以认为是“叶子活动”。但是，不一定只有固定的某一层的活动才可能是叶子，要视具体的活动分层情况而定。叶子活动，才是真正的组成业务活动的基本要素。业务活动之中的参与者，可以认为直

接同叶子活动相关系。

我们可以找出一个基本的业务流程，用叶子活动来说明:

公司销售人员发现新的业务机会;

经过谈判，得到业务机会;

双方草签订单;

公司财务人员决定订单是否有效;

有效的话销售人员查询库存是否足够;

库存足够则双方订立合同;

合同确认则开始发货。

这一个完整的交易流程我们就可以用上面的基本叶子图表示出来。

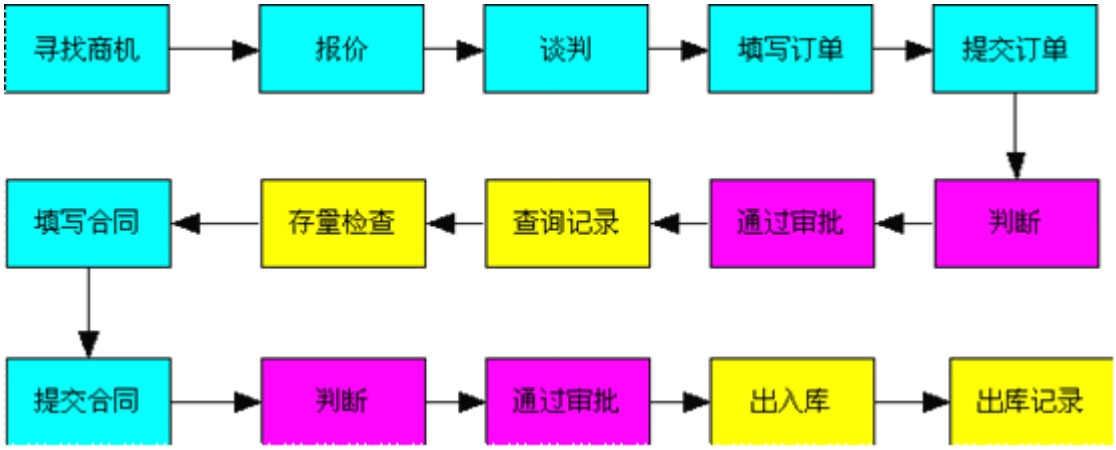


图 14 基本业务流程示意图

3.4 凤凰公司企业信息模型

业务实体（business entity）是企业中的一些起到关键作用的类别。客户、销售人员、订单、合同等等。业务实体是业务模型中一个很关键的因素，在业务流程中几乎所有的业务行为都是直接或间接和业务实体相关的。例如，“销售人员创建订单请求”这个行为成为与订单相关的业务行为。一个完整的业务流程往往是通过多个业务实体相互配合协调完成的，所以我们非常有必要设计这个业务实体行为模型，我们以 ER 图的形式来描述这个业务实体联系模型，并且称之为业务信息模型(Business information model)，在这个信息模型中，业务实体为主导，强调了业务实体间的信息交换，也就是业务实体的行为。

业务信息模型的主要工作是先找出所有业务实体，确定业务实体的属性和行为。要确定业务实体，首先必须确定企业中的角色，并从角色的行为找出业务实体。角色需要我们对目标组织进行讨论。而且在企业中，往往会有一个人担任多个业务角色的情况。从业务角色的行为，我们可以找出业务角色所处理的事物，这些就是我们所需要的业务实体。业务实体是一个独立的业务实体还是业务实体

的一个属性是值得研究的。一个本该是属性的事物被判断成业务实体只是会带来额外开销，可是一个本该单独列出的业务实体却只是被判定为其它业务实体的一个属性就有可能会带来严重的后果，最大的可能是系统难以扩展。

在一个人力资源管理系统CRM中，员工类可能是非常重要的一个业务实体，它可能有非常多的属性。而在其它的系统中，例如进销存，员工类只是起到一个记录、权限管理的作用罢了。再比如，在一些企业内部的自动化处理系统中，客户可能只是其它一些实体的属性，而以客户为中心的设计大行其道的现在，这种设计就有它的致命缺陷。

要确定业务实体的属性和行为，主要是要确定每个业务实体做的事情，属性则是为了能够更好的描述实体和实体要做的事情。

下面的图 15 从总体上描述了我们关心的几个业务组件的业务信息模型。

在这个模型中，按照前面我们提到的业务组件的划分方法分成了 6 个组件：销售模块、销售管理模块、财务管理模块、CRM 管理模块、物流管理模块以及库存管理模块。这些模块的分析以其中的主要人物角色为中心，例如销售模块以销售人员为中心，分析销售人员涉及的主要行为，包括“维护业务机会”、“与客户谈判”等等，于是扩展得到其他的实体内容。而不同的模块之间是有联系的，例如销售人员会向销售管理人员提交报价单，申请批准；这种模块之间的联系也在图 15 中描绘了出来。

在图 15 中，没有把业务实体的属性标记上去，主要强调了各个模块内部以及模块之间的业务联系以及业务行为。在后面我们将逐一分析各个模块的内部实体的行为以及属性。

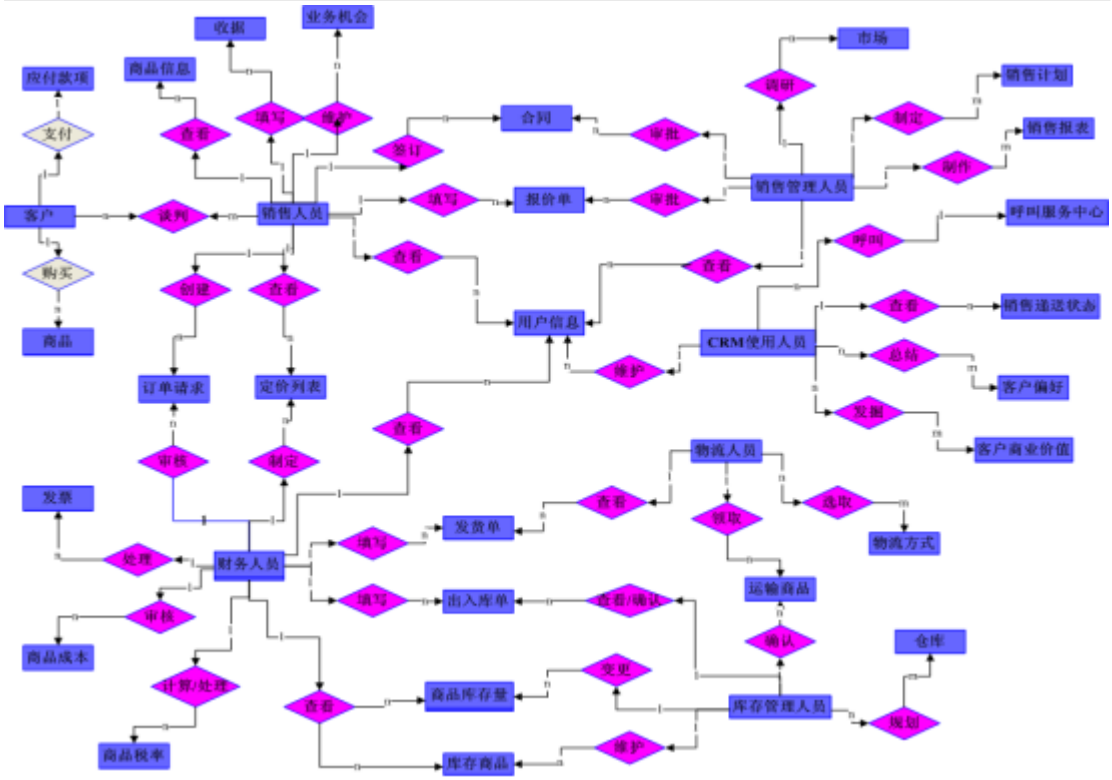


图 15 业务组件整体模型

1、销售模块

主要业务实体有销售人员、客户、订单请求、定价列表、合同、报价单、商品、收据、应付款项、业务机会、商品信息、用户信息等。这些实体间的行为关系以及实体属性如图 16 所示:

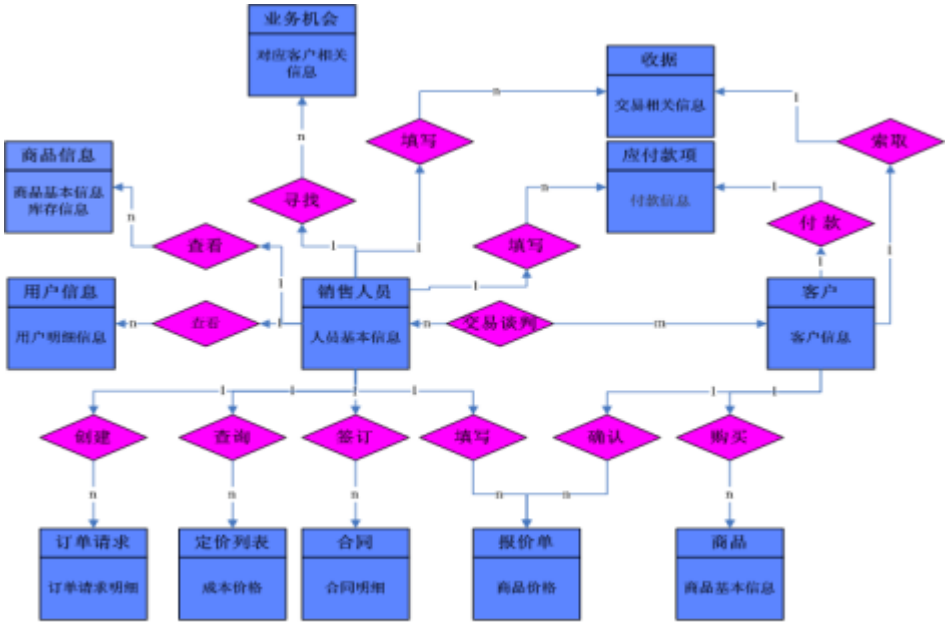


图 16 销售模块业务信息模型

2、销售管理模块

主要业务实体有销售管理人员、市场、合同、报价单、客户信息、销售报表、销售计划等。这些实体间的行为关系以及实体属性如图 17 所示:

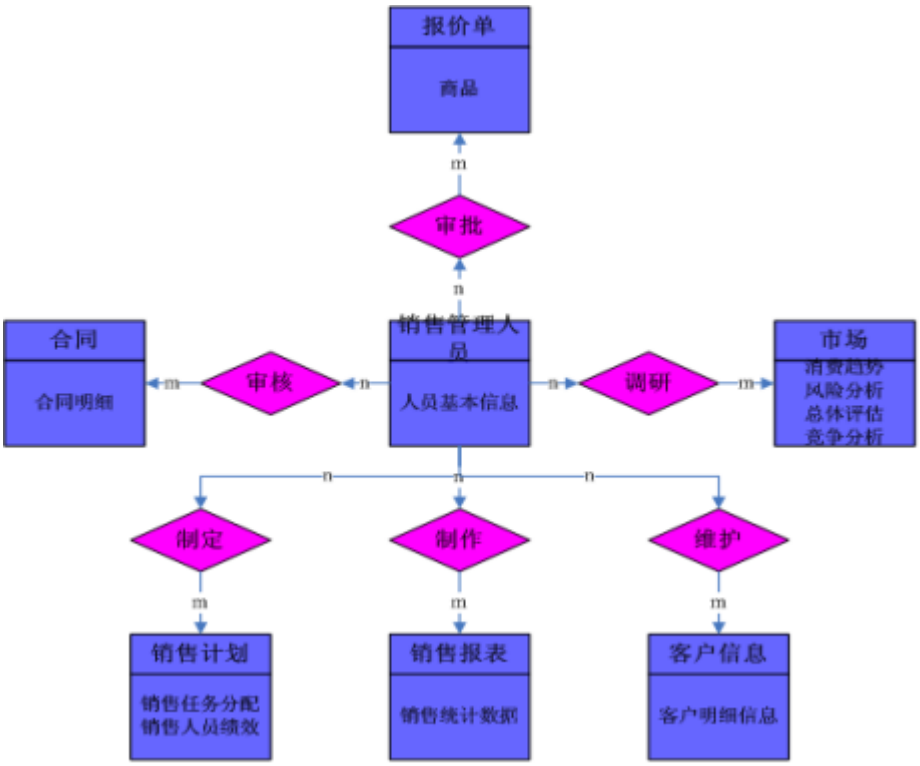


图 17 销售管理模块业务信息模型

3、财务管理模块

主要业务实体有财务管理人员、库存、客户信息、订单请求、商品成本、税率、商品价格列表、发票、出入库单、发货单、订单存根等。这些实体间的行为关系以及实体属性如图 18 所示:

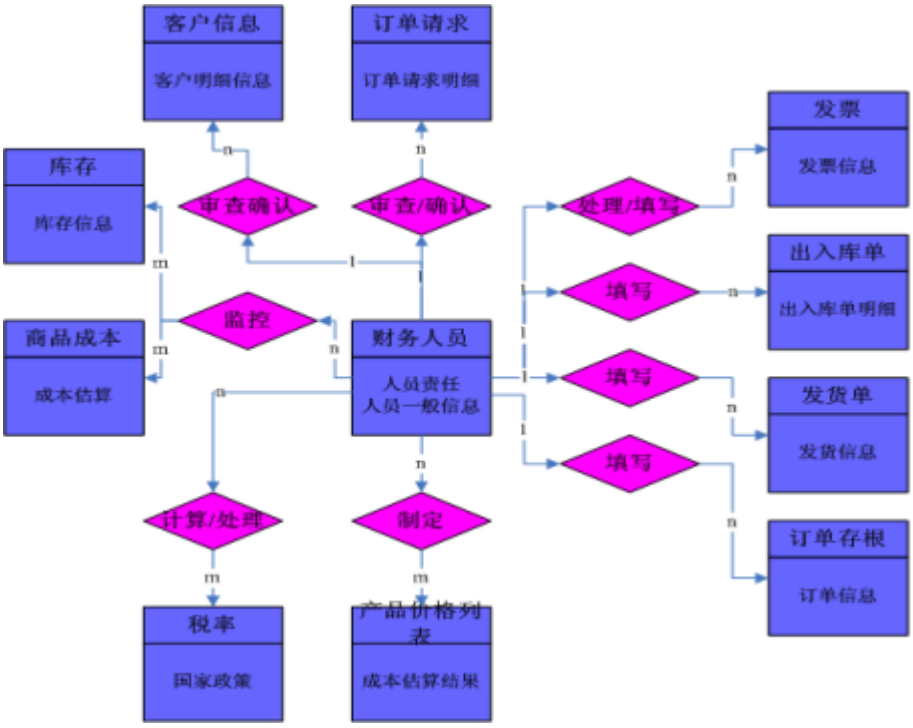


图 18 销售模块业务信息模型

4、CRM 人力资源管理

主要业务实体有 CRM 管理人员、客户信息、销售/递送状态、呼叫服务中心、用户、客户偏好、客户商业价值等。这些实体间的行为关系以及实体属性如图 19 所示:

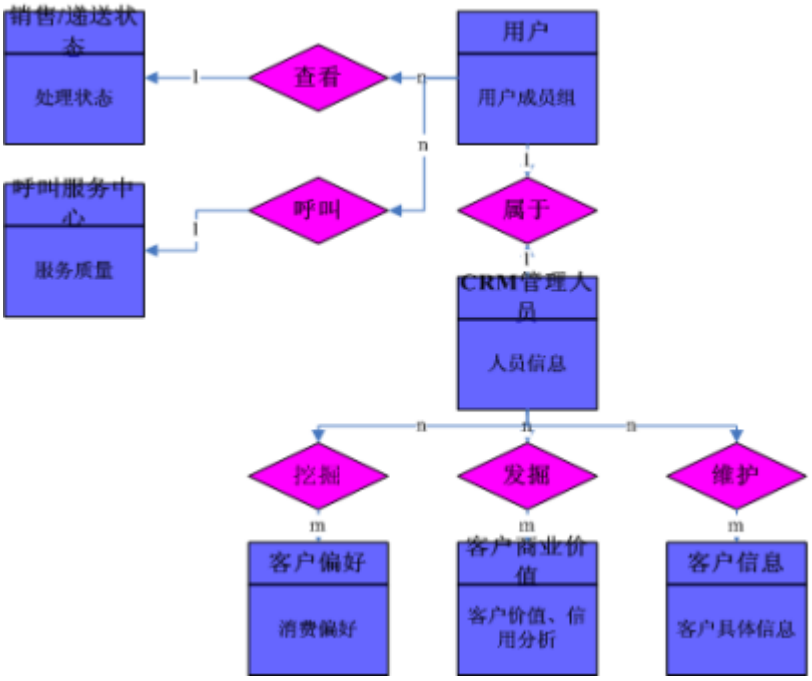


图 19 CRM 人力资源模块业务信息模型

5、库存管理模块

主要业务实体有库存管理人员、库存信息、库存量、出入库单、仓库等。这些实体间的行为关系以及实体属性如图 20 所示:

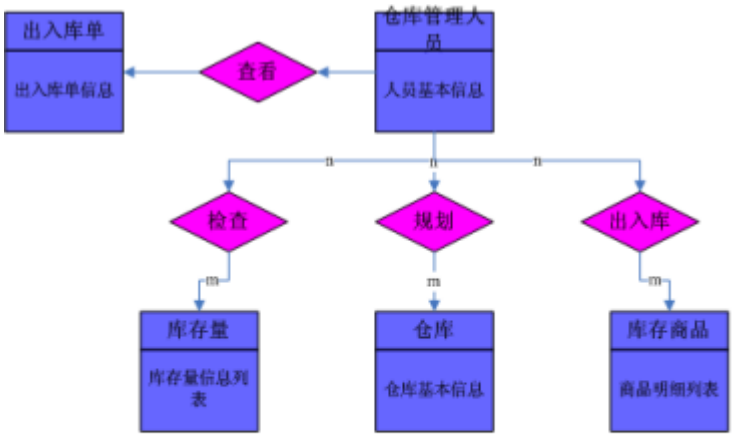


图 20 库存管理模块业务信息模型

6、物理管理模块

主要业务实体有物流管理人员、发货单、运输商品、物流方式等。这些实体间的行为关系以及实体属性如图 21 所示:

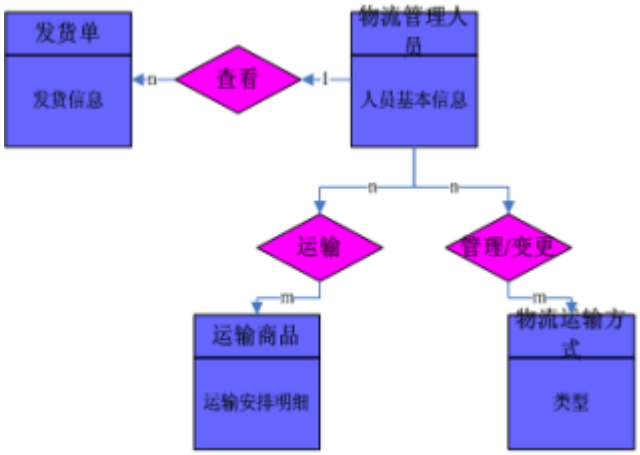


图 21 物理管理模块业务信息模型

3.5 凤凰公司活动／信息矩阵

活动和信息的视角是两个最重要的视角，它们之间的关系也非常紧密，一个业务活动的名字通常是一个动－名形式的词组，其中的名词通常就是信息模型中的实体。为了表示这种关系，我们用活动－信息矩阵来刻画信息的使用关系。

凤凰公司活动－信息 CRUD矩阵

事件 \ 信息	商品信息	用户信息	定价列表	业务机会	订单请求	报价单	合同	应付款项	市场信息	销售计划	销售报表	处理状态	客户偏好	商业价值	发票	收据	出入库单	发货单	库存量	税率	递送方式
调查市场信息									CRU				CRU	CRU							
寻找潜在用户		R		CRU									R	R							
分析客户价值									R				CRU	CRU							
联系客户		CRU							R				R	R							
建立新客户账号		CRU																			
查询老客户信息		R																			
创建业务机会	R	R	R	CRU																	
管理业务机会	R	R	R	CRU	CRU	CRU	CRU														
提供销售方案	R		R			CRU															
确定产品种类和型号	R																				
确定订货数量	R	R	R										R	R							
查询库存量是否满足	R																		R		
报价		R				CRU														R	R
查询产品价格列表			R																		
计算折扣率	R	R												R							
计算利润率						R														R	
审批报价方案					CRU	CRU															
签订合同	R	R	R	R			CRU														
收取预付款								CRU							CRU	CRU				R	
验证客户信用度		R												R							
生成销售订单	R	R	R	R	R	CRU															
通知相关人员	R																				
审查订单请求					RU																
调度出货方案																		CRU			
填写出货单																	CRU				
填写送货单																		CRU			
收款								CRU													
开发票及收据															CRU	CRU					

图 22 活动信息矩阵

4. 遗留系统分析

我们的遗留系统是 ERP 和 CRM 系统，这两个系统覆盖了我们期望存在的大部分组件和功能，为我们的工作提供了良好的基础。不论是对于组件的实现还是服务的设计，我们都需要对遗留系统进行详尽的分析。

对于 ERP 系统，我们将它分解为库存管理子系统、成本管理子系统、物流管理子系统、财务管理子系统等主要子系统；另外，CRM 系统中划分了人力资源管理子系统、业务机会管理子系统、人员通知子系统，这些子系统将成为服务实现的主要来源。

对于遗留系统的分析，将主要在我们的实现阶段进行。

5. 转换实施路线图

我们使用下面这个实施路线图来描述我们对于 SOA 实施的想法：

	Short term	Mid term	Long term
Solution	Through integration of disparate data, significantly reduce processing and transaction cycle time throughout the enterprise and speed claim resolution	Expand inter-enterprise collaborative capabilities through increased warranty data sharing (e.g., detailed claim data, part/subsystem interfaces and part interoperability)	Orchestrate business events in an on demand fashion through full integration of business rules among OEM, supplier and dealer
Technology evolution	• Collaborative applications to integrate data repository (avoid duplication of data) • Normalization of data for data interpretation	• Portal capabilities for OEM/ supplier data sharing • Workflow engine with event triggers • Optimized use of text mining tools	• Service oriented integration of infrastructure, data management and analytical capabilities • Grid/utility service for variable infrastructure model • Autonomic technology to support analytical capabilities
Benefits	• Compressed time to process claims resulting in fewer incidents for a given problem	• Contained warranty reserves as a percent of revenues • Improved accuracy of claim data to facility cost recovery with suppliers • Reduced parts inventory associated with fewer claims	• Improved quality management through module/system integration versus component performance • Warranty performance integrated to supplier overall performance • Improved customer satisfaction and brand impact • Variable investment in warranty application management solutions

图 23 转化实施路线图

图中分别阐述了 SOA 的短期、中期、长期计划，这个也会和我们将要提到的 SOA 成熟度模型结合起来，以便于在不同时期了解实施情况以及及时修订实施方案。

6.电子商务模式

在这个 SOA 实现过程中，我们可以利用电子商务模式的概念帮助我们分析解决问题。电子商务模式是一组已被证实的、可重用的资源，它有助于加速电子商务解决方案的开发过程。我们把电子商务模式的概念框架应用到这个实现里面，可以让我们的解决方案更加符合现实中电子商务的要求。

参照了 IBM 的电子商务模式思想，我们使用下列步骤来考虑业务实现方案:

第 1 步：规划高级业务描述

我们需要考虑的主要流程是订单处理。我们的用例分析中清楚的描述过订单流程工作的内容，订单处理流程是一个相对复杂的流程，下图粗略的描述了相关的人、流程、工作:

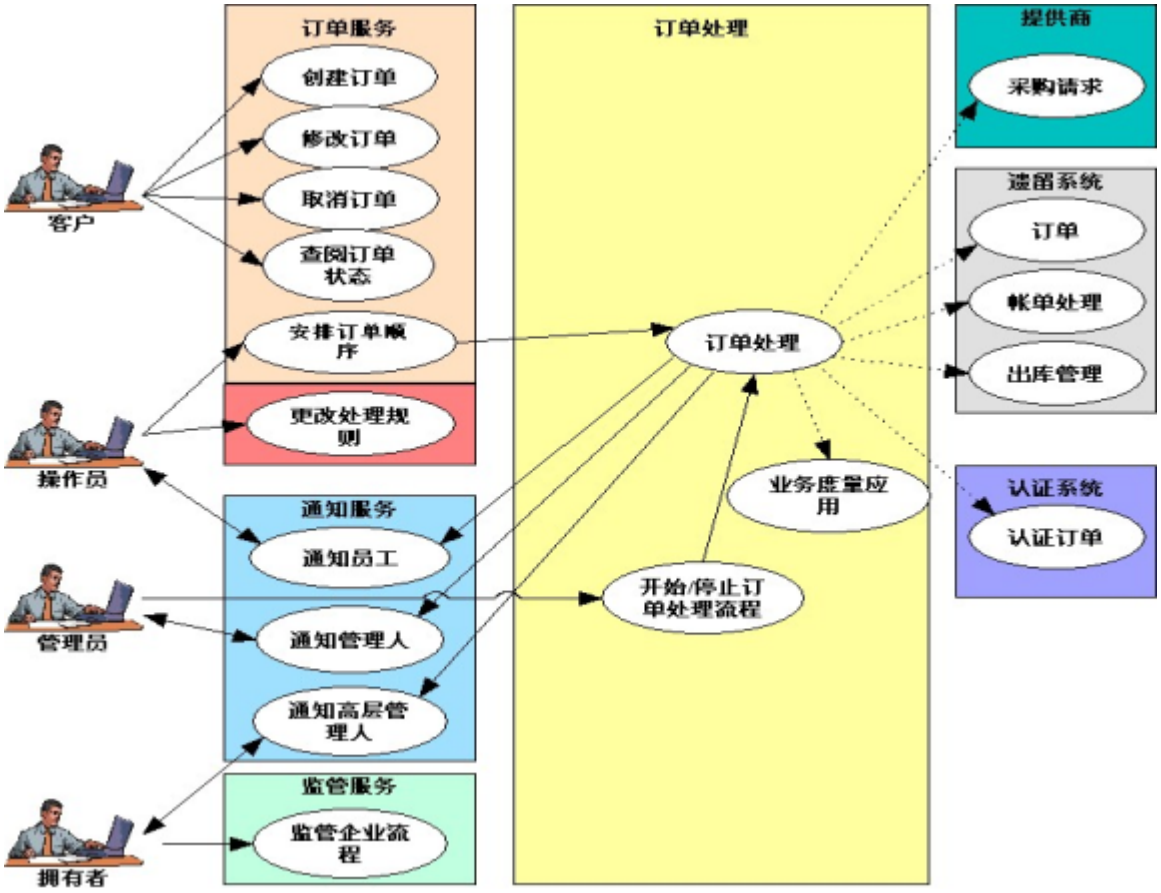


图 24 订单处理流程分析图

这个图给我们大致描述了与订单处理相关的内容，这些内容能够反映出许多与订单处理流程相关的问题。

第 2 步：设计解决方案概要图表

通过将业务描述中的文本转化为图示的方法生成一个解决方案概要图表。我们在用例分析的过程中完成后进行了这项工作。我们使用不同的用例来生成解决方案概要图表，对应上面的业务描述图，得到下面的解决方案图:

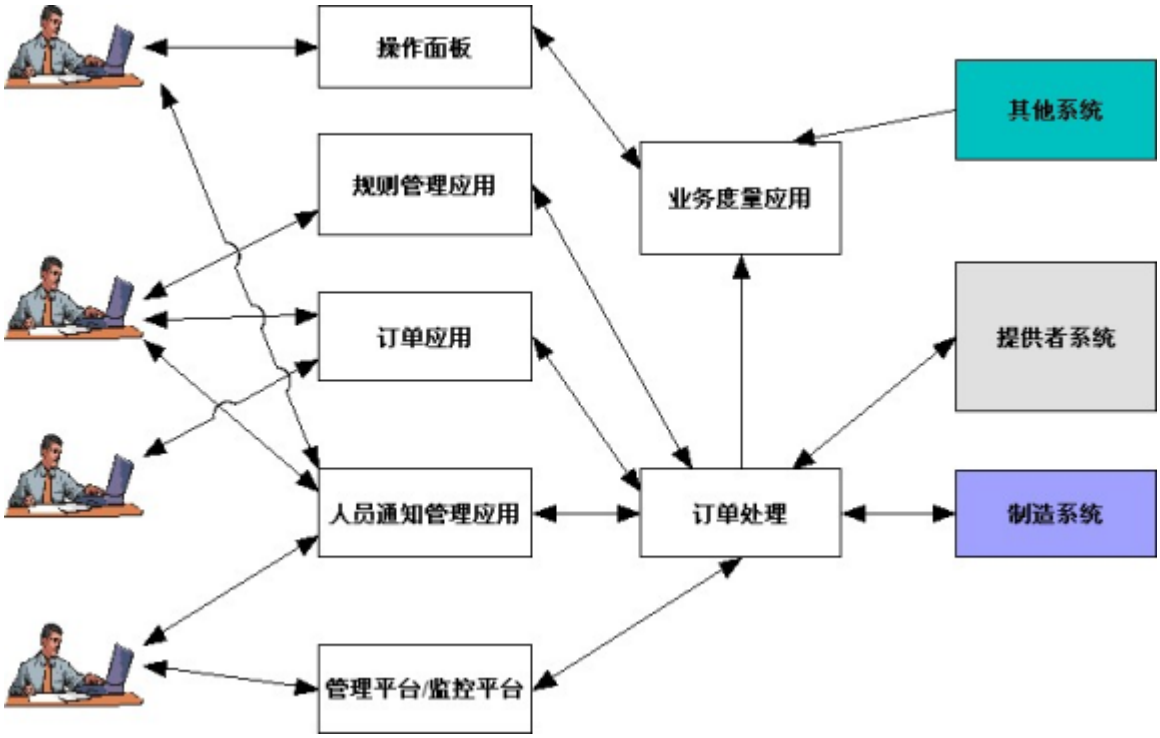


图 25 解决方案图

第 3 步：识别业务模式:

业务模式是用于确认用户、业务、数据之间的联系的。所以业务模式用于端对端的应用程序，下面列出一些识别出的业务模式:

操作面板允许持有者通过查看来自于不同系统的聚合信息来监视业务运作。这就是信息聚合（Information Aggregation）模式，这是业务模式的一种。

在业务流程中，操作员可以:

· 建立订单
· 查看订单状态
· 取消订单
· 更改交易规则

而管理员可以:

· 启动定制的流程
· 停止定制的流程
· 将定制的流程挂起
· 中断后继续定制的流程

这些操作包括指导用户与后端系统（预定处理系统）进行交互，并指定一个自服务（Self-Service）的业务模式。流程中分别通知操作者、管理员、用户任何在操作上的、系统中的，以及交易过程中出现的问题，然后管理相应问题的解决方案工作流程，其中包括对等协作。这些交互说明了协作业务模式，协作业务模式也是业务模式的一种。

我们需要一个供给链来向供应商发送购买订单来对订单及其实现进行进一步的处理。这种交互在企业之间进行，并说明了扩展的企业业务模式，这也是业务模式的一种。

第 4 步：识别集成模式

使用单一的签名(SSO)来访问不同的应用程序这种功能的实现需要得到所有用户的共识。为了实现这种功能，需要访问集成模式。

从前面解决方案的概要图表中可以明显地看出，流程集成模式（应用程序集成模式的子集）可以使得这个订单系统链接到不同的应用：

· 业务度量应用程序

· 通知处理器

· 规则管理应用程序

· 供应商的系统

· 用于生产的应用程序

通过业务度量应用程序链接到其它系统，业务度量应用程序集合了从订单系统及其它的系统发出的处理信息。也就是说，需要数据集成模式（另一个应用程序集成模式的子集）。从这些方面我们可以识别出集成模式。

第五步：识别复合模式

至此为止识别过的模式，在下图中予以总结。但是我们还没有找到适应于订单处理系统需要的复合模式。

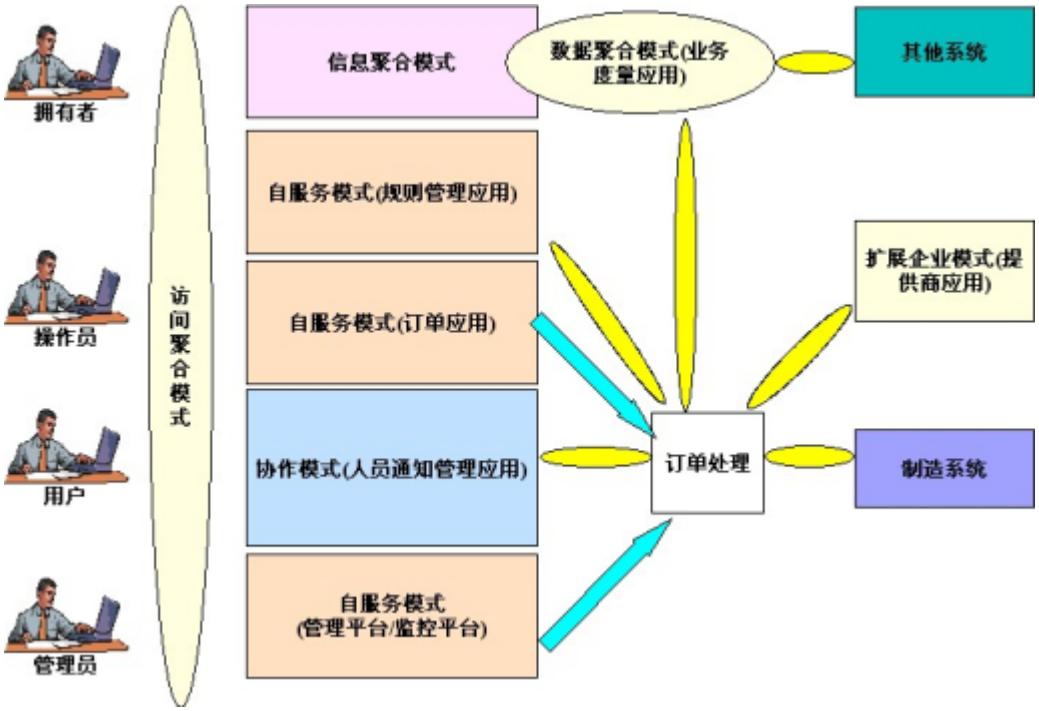


图 26 已经识别出来的各类商务模式图示

第 6 步：识别各种应用的模式

针对每一种应用，我们还有对应可以选择的应用模式，在实现中，我们需要选择这些应用模式。

6.1：自服务

由于只关注包含后端集成的 Web 通道，所以直接集成的单通道是订单入口、规则管理，以及管理控制台等应用程序宜采用的模式，即自服务模式应用都是选择的直接整合单通道模式。

6.2：协作

一旦收到需要通知应用的消息，就应当密切关注操作员、管理员或持有者的行为，并允许他们调换工作。换句话说，需要通知处理器的支持，不仅用来向用户通报，也管理相应的对等工作流程。因此，宜采用受管理的协作模式。即这个协作模式的应用程序模式为受管理的协作模式。

6.3：信息聚合

只要对聚集的业务过程的读取访问感兴趣，就需要用户信息访问应用程序模式。到目前为止，还不需要用户存储结果的本机副本或者执行对源程序的修改。因此，用稳定的基本类型就可以了。

6.4：扩展的企业

我们需要集成不同的供应者以及替换供应者，只要他们的信息格式兼容就无需更改系统配置。换句话说，我们需要按照 1：N 的比例来设计提交给大多数设备的请求。Exposed Broker 公共路由模式的路由器变量是最能符合这些需求的。

6.5：访问集成

我们需要一种共识使得不同类的人能够使用单点登录(single signing on)对不同的应用程序进行访问。换句话说，我们需要表示性的安全服务。单点登录应用程序模式已经提供了这种安全服务。我们可以通过使用直接集成单通道（Directly Integrated Single Channel，DISC）应用程序模式（一种自服务的应用程序模式）中的变量来实现表示性的服务。应用这种模式的主要驱动程序提供了对后端应用程序以及 Web 应用程序中的数据进行实时访问的功能，它以“后端应用程序和数据”作为主键。但是我们可以使用它通过单一的 Web 应用程序来访问多个 Web 应用程序，这样就提供了一个公共的表示层。把变量并入最初的 DISC 应用程序模式中是暂时的，并且它仍旧作为直接集成单通道应用程序模式。我们把这个变量看作页面聚合应用程序模式它仅代表 Web 应用程序的集合，也就是不包括自服务模式。

6.6：应用程序集成

应用程序集成包括流程集成和数据集成。

6.6.1 流程集成:

直接联结的应用程序模式可以用来记录应用程序与应用程序之间，系统与系统之间以及应用程序同其它系统之间的交互过程。

直接联结的应用程序模式可以用来将通知处理器与整个系统集成起来。

6.6.2 数据集成:

由于业务度量应用程序只是聚集了那些来自于不同数据源的数据，所以用 Population 应用程序模式就可以实现了。

6.7 识别了所有的应用模式后，得到下图形式，对应上面的图:

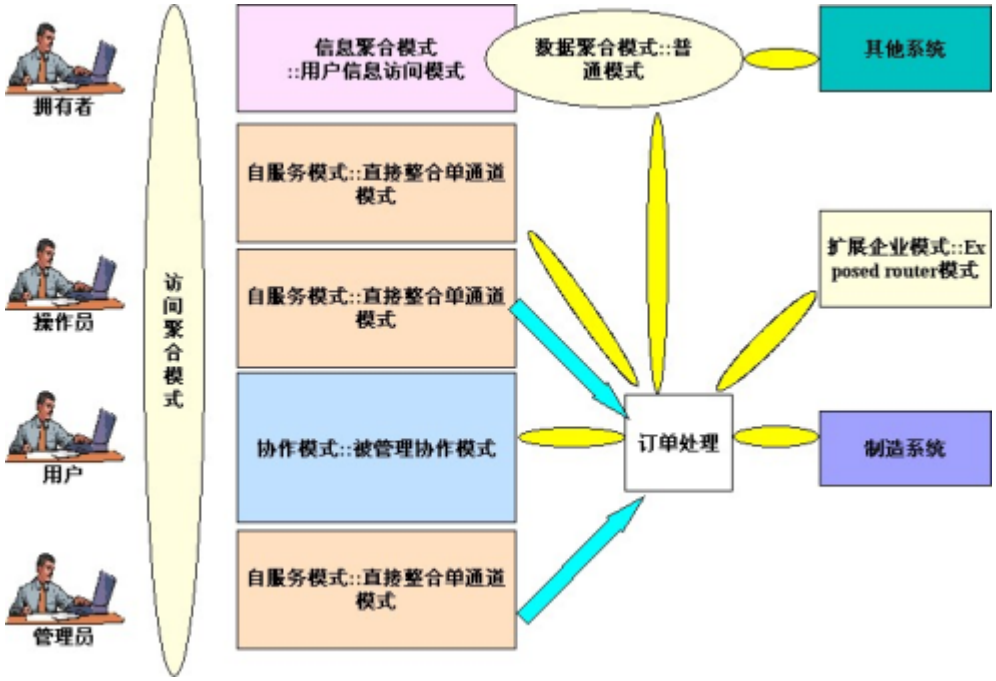


图 27 应用模式识别图示

第 7 步：将程序包集成到解决方案

我们可以使用公共事件体系结构（Common Event Infrastructure，CEI）来聚集我们系统及其它的系统发出的流程事件。CEI 提供了数据集成：Population 应用程序模式，并且通过使用流程集成：直接联结的应用程序模式来支持集成。CEI 没有提供直接访问数据库的事件，但它提供了 API 用来查询数据库。换句话说，它可以被视为一种数据服务

第 8 步：识别运行时模式

在应用程序模式选定的基础上再选择运行时合适的模式。

8.1 对于访问集成模式中的单点登录和页面聚合模式

无需使用不同种类的应用程序服务器，也就是说，应用程序将运行在同类的应用程序服务器上。这样，对于 SSO，可以使用同类应用程序服务器运行时模式，如下图所示:

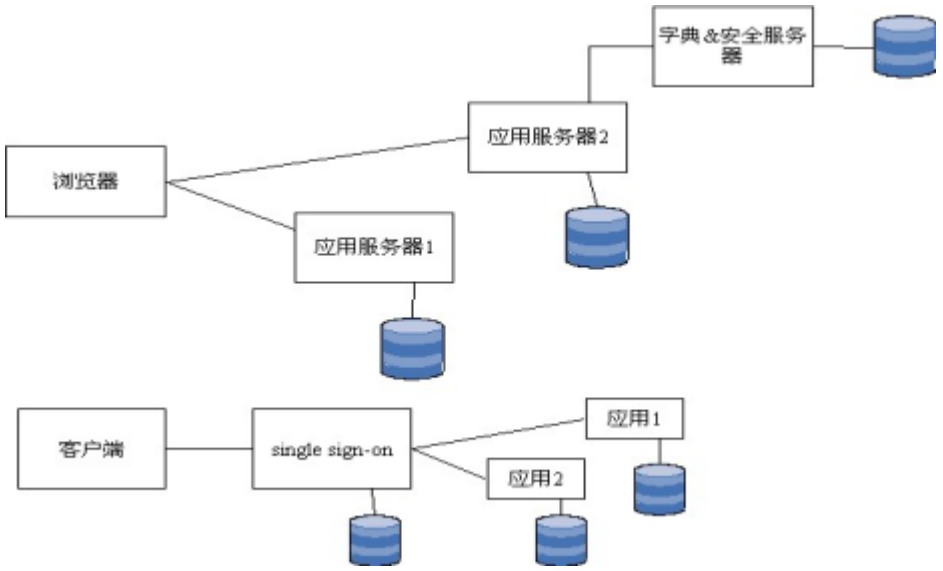


图 28 访问集成::单点登录::同类应用程序服务器模式图

这里表示的就是访问集成::单点登录::同类应用程序服务器的模式。

单点登录模式使用同服务器运行模式，将不同的应用程序应用在同类的服务器上；而对于页面聚合的模式，使用页面聚合运行时模式，如下图所示：

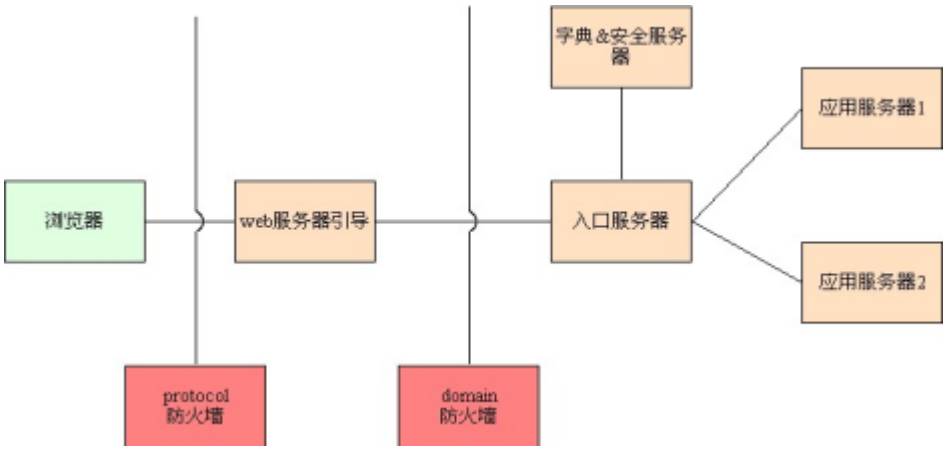


图 29 访问集成::页面聚合::页面聚合运行时模式

8.2 用户信息访问模式

我们的访问模式中，不能直接访问数据源，用户信息访问运行时模式仍然存在，如下图：

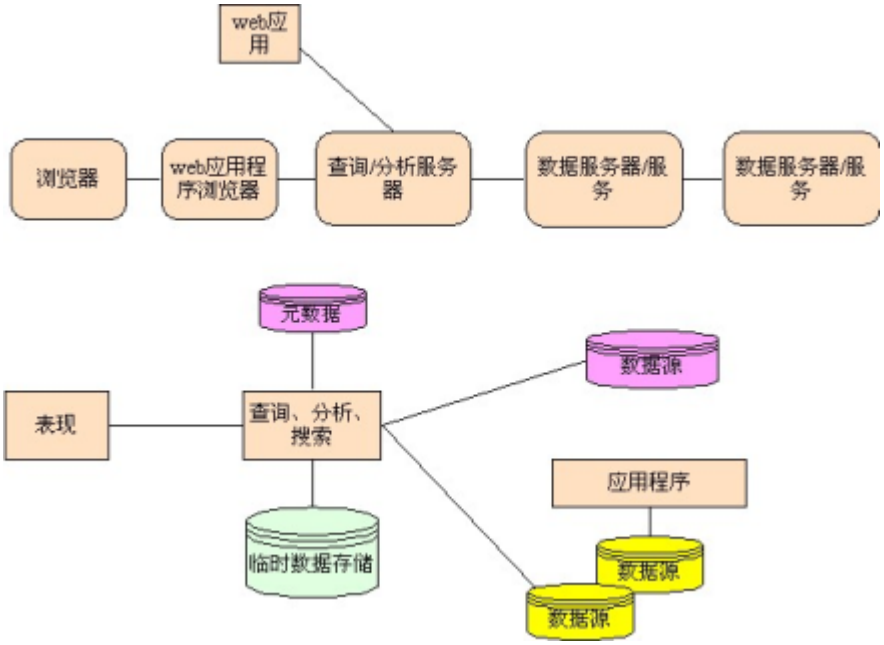


图 30 只读的信息访问::在内部网络中的 BI 应用程序

8.3 直接访问的单通道模式

鉴于电脑黑客的攻击能力不断加强，如下图示的运行时变种 1 是适合于直接集成单通道应用程序模式的一种值得推荐的模式。

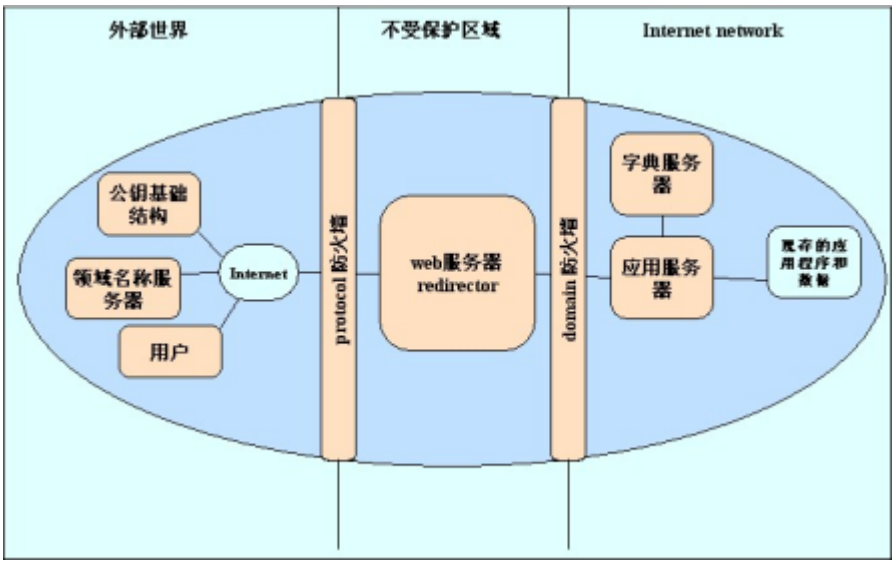


图 31 自服务::直接集成单通道::运行时变种 1

8.4 管理协作模式

在管理协作模式方面唯一一种类型是电子商务模式一书中介绍的应用程序模式。因此，我们引入并使用了一个潜在的新模式——管理协作运行时模式。

管理协作运行时模式，如下图所示，允许 Web 客户端之间可以进行同步或异步的合作，也就是说，接收端不必通过预定义的方式连到协作服务器上。它通过处理从预定义的工作流程中得到的请求并由接收端存储随后的修复响应来实

现了这个功能。

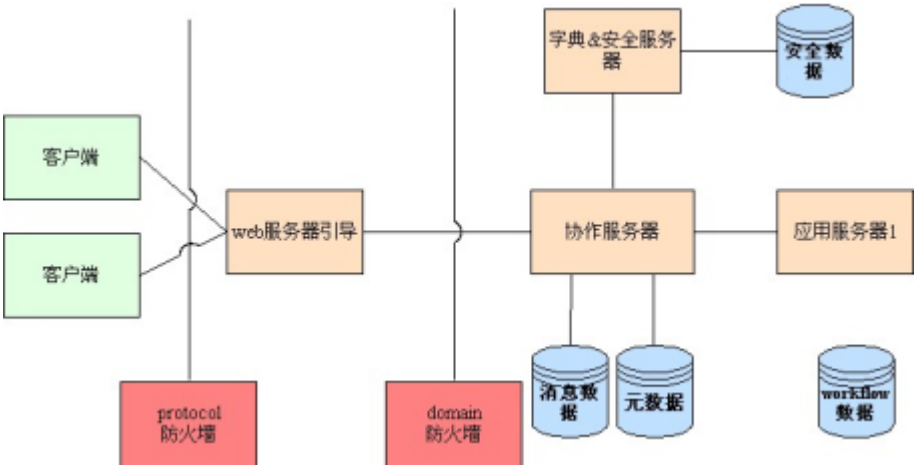


图 32 管理协作运行时模式

协作服务器在处理工作流程同客户端的交互过程中起了重要的作用。它为使用流程数据信息的实际用户设计了工作流程信息接收器（例如，用户、管理员、组织、任务等等）。它也将负责存储客户端的工作流程信息并允许他们查看并回复那些信息。一旦收到回复，服务器就会将它传递到工作流程中并删除原始的信息。

工作流管理负责管理已安排好的工作流程的执行过程。它将维护工作流程的状态直到执行完毕，也负责将接收到的信息映射到工作流程的实例中去。

8.5 直接连接模式

如果我们想将不同的组件用于服务，那么使用 Simple Service Bus 模式。这也是 ESB 总线的思想。

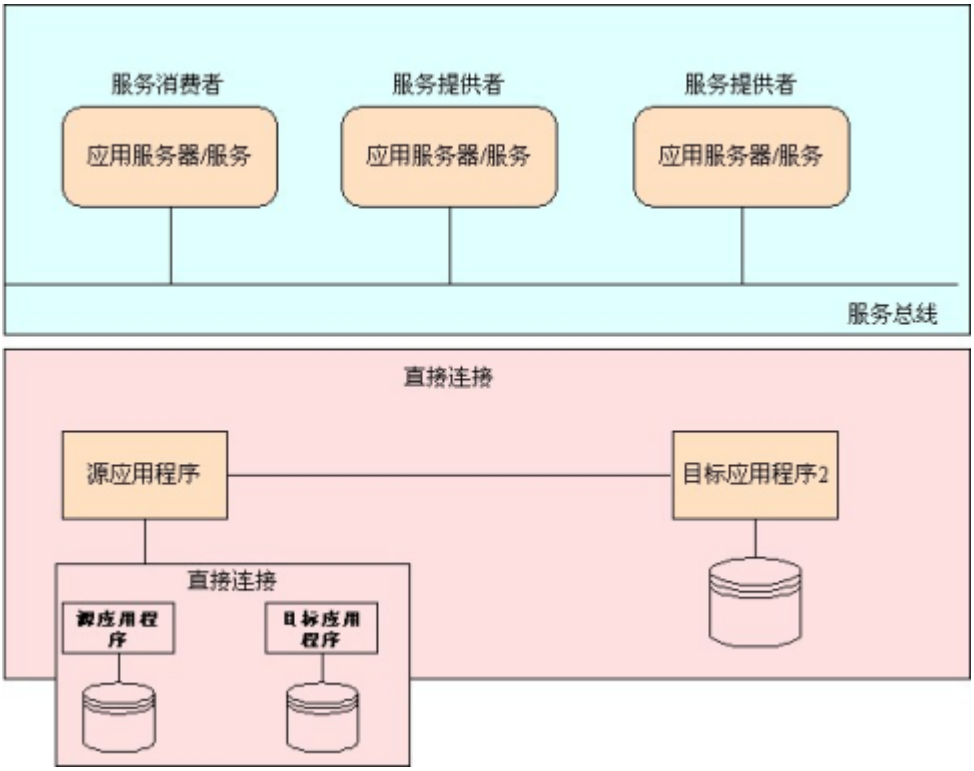


图 33 直接联结::Simple Service Bus 模式

8.6 串行模式

串行流程运行时模式，如下图所示，允许系统以串行的方式与不同的应用程序交互。这个交互由流程管理员管理。

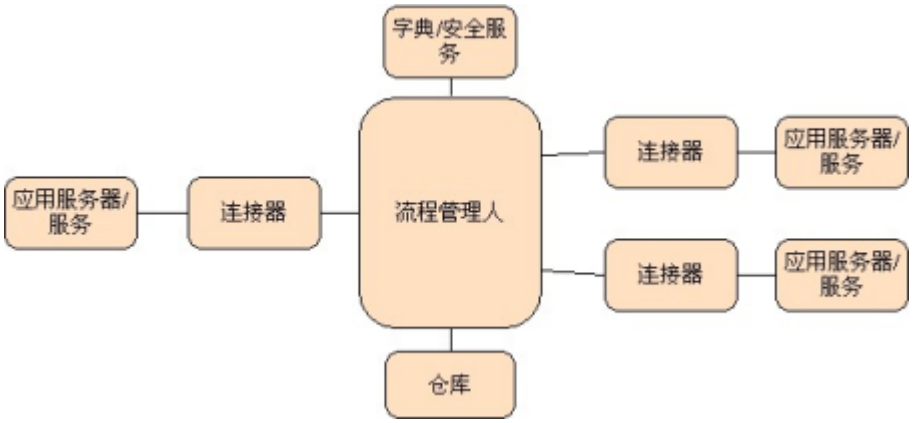


图 34 串行流程运行时模式图示

8.7 公共路由器应用程序模式的运行时模式

我们需要一个与用来集成不同供应者的公开路由器应用程序模式进行通信的运行时模式。它应当支持 1: N 的设计规划，并支持转换成用于访问现有系统的协议。公开路由器运行时模式或公开企业服务总线（ESB）模式能够满足那种需求。建立一个 SOA 解决方案，公开的 ESB 模式对于您来说是个较好的选择，它提供了一套底层功能使得能够成功集成 SOA 中的服务。

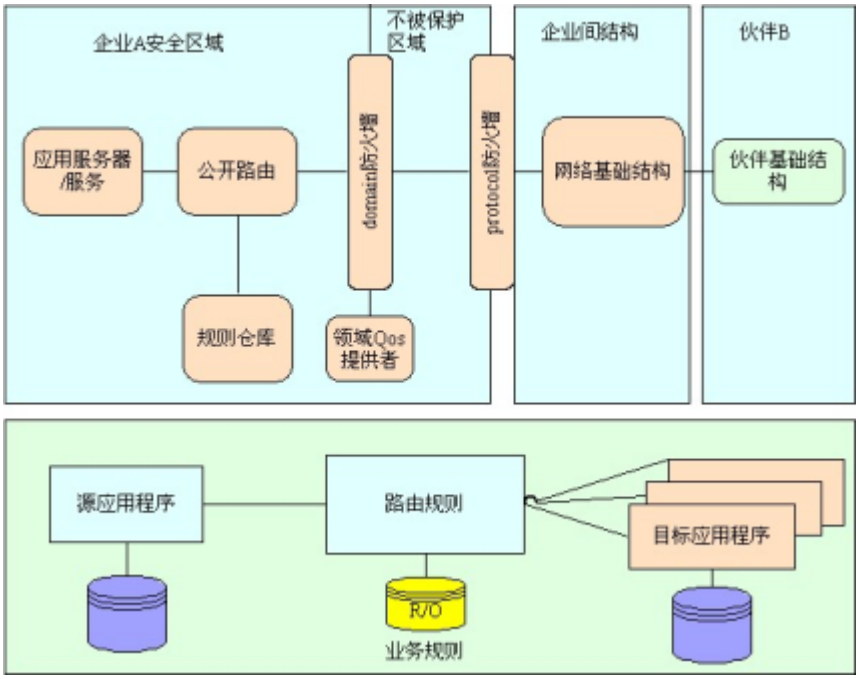


图 35 (Exposed Broker:: Router 模式)

公开路由器运行时模式图示

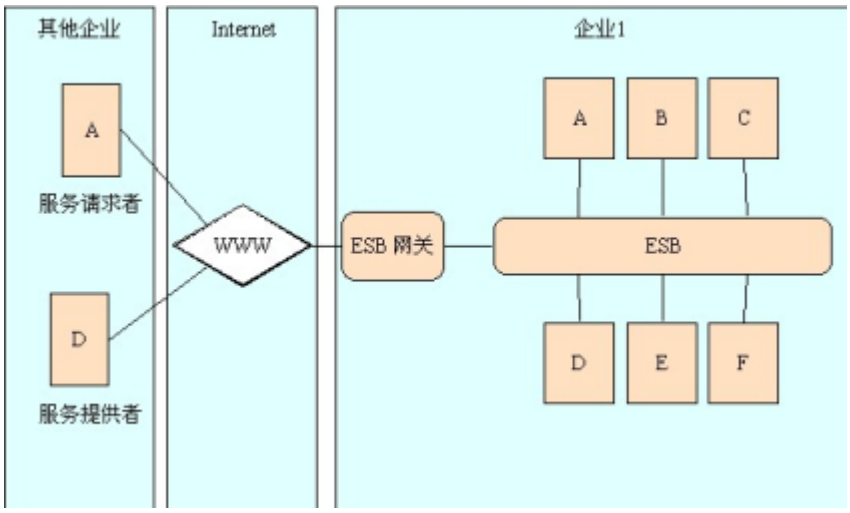


图 36 Exposed ESB 网关复合模式

8.8 所有模式总结

在经过上面所有的运行时模式识别后，我们得到的模式总结如下图所示。注意 Simple Service Bus 和 CEI 模式都是 ESB 模式的一部分。同样地，流程管理器和工作流管理器已经并入了单独的流程管理节点，因为流程管理也有管理工作流的能力。

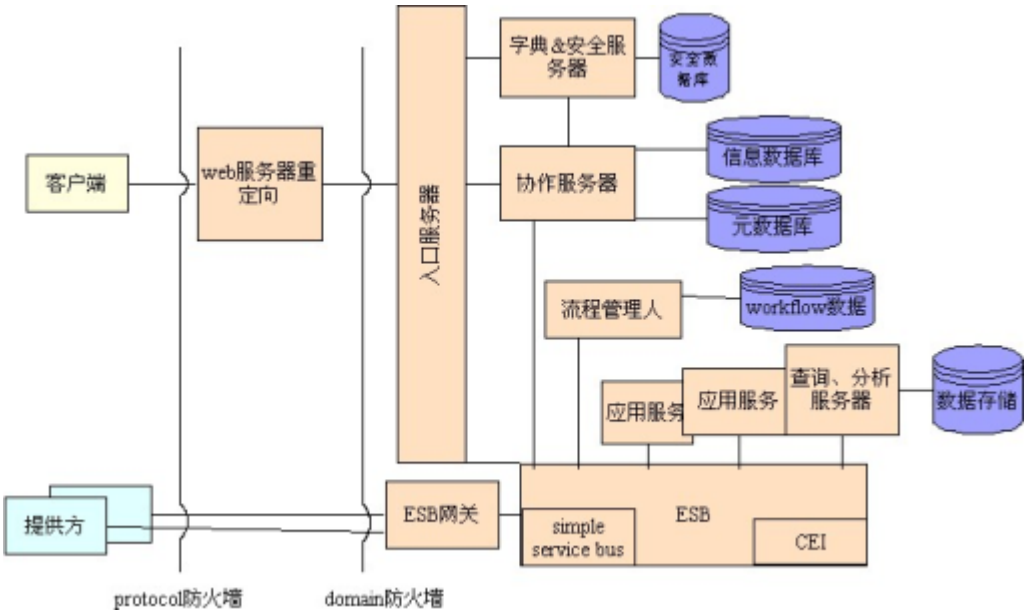


图 37 运行时模式总结图示

上面一些内容基本阐述了我们对于电子商务模式在我们现在希望实现系统中的可能体现，在实际开发中可能会有各种各样的问题，从模式的角度出发考虑，能有一个整体的把握。

对于电子商务模式的应用是一个从顶至下的过程，从业务模式到运行时模式是对系统相关功能逐步细化的过程。另外，电子商务模式本身仍是在研究中的内容，所以以后能发展成什么样，以及能够如何的协助企业业务的实现，都是我们关注的问题。

7.企业 SOA 成熟度分析

在实施 SOA 之前，需要有一种合理的评价方式来帮助 IT 决策者们对 SOA 实施和规划的战略价值进行测量与评估。SOA 成熟度模型（SOA Maturity Model）可以为 IT 和业务用户提供一种框架，使其能够正确地评估 SOA 在企业中的适用性和收益。使用 SOA 成熟度模型分析现有的体系架构，可以迅速地为 SOA 项目构建远景、范围和规划，还能确定成功的关键性能指标。

在这种情况下，众多厂商都提出了 SOA 的成熟度模型，其中 BEA 和 IBM 各自有一套评价体系，并且有各自的工具支持，我们在分析 SOA 成熟度的过程中，主要参考了 IBM 的成熟度评价体系。

下图显示了一个基本的成熟度评价模型：

图 38 成熟度评价 5 级模型图

在这个评价体系中，主要的评价依据来自于 EA 团队的建设情况。对于 EA

建设活动来说，主要考虑的是两个方面，如下图所示，一个是 Business Architecture，一个是 IT Architecture。

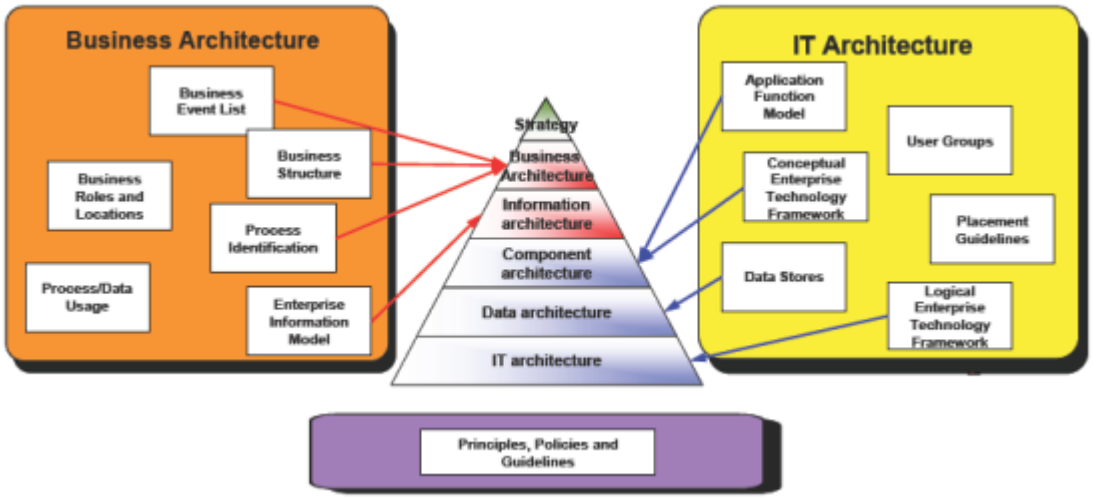


图 39 企业 EA 层次图

第 1 级是初始化级别。第 1 级的组织通常没有正式的体系结构流程。体系结构没有从项目分离出来。通常，这些组织不具有 EA 团队；每个项目团队通常根据 LOB 划分，彼此独立地进行工作。精力主要放在交付单个项目上。

第 2 级是可重复级别。在此级别，进行了一些体系结构方面的工作。项目团队通常定义一个可重用体系结构，在多个项目间使用。项目团队之间建立了非正式的通信渠道。一个 EA 团队将帮助在较为混乱的环境中形成结构，促进项目团队间的通信；不过，在此阶段，仍然很少存在此类团队，通信是临时性的，较为混乱。

第 3 级是已定义级别。这个级别的组织在 EA 活动方面进行了一些投资。配备了 EA 团队，为其指定了对体系结构元素进行标准化的任务，负责进行创建参考体系结构的活动，就此体系结构对项目团队进行培训，并定义控制和执行策略。通常，EA 团队将创建一组技术组件和框架，然后标准化各个项目团队间对这些框架的使用。

第 4 级是已管理级别。当 EA 团队开始定义 SOA 路线时，就达到了这一级别的成熟度。今天，每个大型组织都有一群架构师在谈论 SOA。最起码的，这些架构师看起来已认识到 SOA 的价值，并在尝试形成 SOA 策略。如果组织的 SOA 活动主动参与为 LOB 服务的项目团队的工作，则此组织可归到第 4 级。项目团队和 EA 团队需要进行协作，以定义组织的 SOA，包括流程、技术和组件。应该定义控制和“奖励”策略。需要建立支持级别，且要清楚地了解何时联系某人以及进行联系的原因。必须配备分析人员用来定义服务的框架。

第 5 级是优化中级别。这个级别体系结构流程和策略都已制度化。对服务价值有了清晰的认识。配备了框架，供每个团队公开和使用服务。在此级别，组织

可以真正地充分利用 SOA 的价值。他们开始了解如何与其业务合作伙伴、供应商和客户交换服务。为了实现最大业务灵活性，业务服务级别的重用（不限于技术组件）成为了体系结构的核心。在此级别，组织将看到拥有可以迅速响应业务需求的灵活 IT 组织的成本优势和时间优势。

应该可以看到我们需要达到第 5 级的优化中级别，才能真正看到 SOA 带来的随需应变的理念。而我们在 SOA 实施过程中，可以通过这个成熟度模型来分析我们现有的架构是否相对以前的架构有成熟度的提升或是退步。

我们利用上述成熟度模型分析现在凤凰公司的 IT 体系架构，这里的基本架构情况是由我们自己设想的，在实际项目中，我们应该利用咨询部门得到我们实现成熟度评估需要的资料。

我们在我们提到过的业务组件分解的基础上进行成熟度的评估，前面已经详细陈述过。在 CBM 业务组件建模的基础上，我们对于每个业务组件模块进行成熟度分析得到结果。在原有 ERP 系统的管理下，我们认为凤凰公司的库存管理部门、采购部门、物流部门都是拥有 EA 团队，创建一组技术组件和框架，然后标准化各个项目团队间对这些框架的使用。所以我们评价他们为已定义级别的成熟度。

同样的道理，而在原有 CRM 系统的管理下，我们认为凤凰公司的人力资源管理部门、售后服务管理部门都是已定义级别的。

而我们关注的重点部分是销售人员，销售管理人员，以及财务人员，在这里，我们认为他们进行了一些体系结构方面的工作，项目团队通常定义一个可重用体系结构，在多个项目间使用，项目团队之间建立了非正式的通信渠道。通信是临时性的，较为混乱。这一点主要体现在 CRM 和 ERP 系统的信息整合问题上，ERP 和 CRM 系统各自维护一套数据，存在数据冗余问题的同时，在数据通信上也存在问题，销售人员的效率还需要加强，所以他们的成熟度等级评价为可重复级别。

因此，我们在实施 SOA 项目过程中的关注点就直接定在销售部门和财务部门，通过实施 SOA，使得 SOA 成熟提升，到达 5 级水平，真正实现可以迅速响应业务需求变更，实现 SOA 实施的目标。

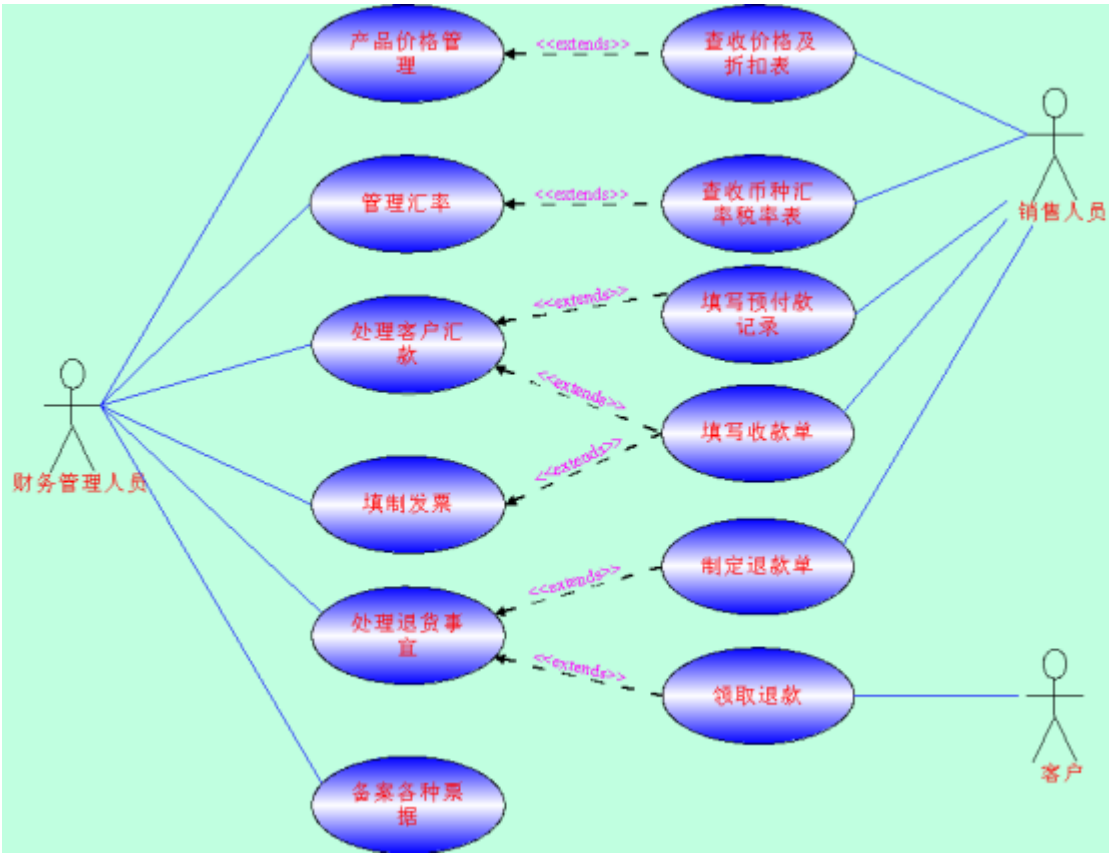
8.总结

业务模型分析设计文档针对凤凰公司的业务相关问题，从多种角度进行了分析。文档对业务组件模型，企业架构，电子商务模式和 SOA 实施相关方面问题都进行了详细的阐述，是所有工作的基础，我们在后续工作中的许多思想也在这个文档中体现出来。在业务建模的基础上，我们将接下去进行服务和组件相关的工作，包括服务的发现、定义、实现以及组件的接口定义、实现等。我们的分析

工作是按照一定顺序进行的，在业务模型分析的基础上逐步精化对应的内容。

系统用例描述

1.财务管理人员相关用例



1．管理税率

用例名称	管理产品价格
标识符	UC_Finance1
用例描述	财务人员对产品的价格进行管理
参与者	财务管理人员，销售人员
基本事件流	1 财务管理人员输入用户名和密码，登录财务管理子系统 2 查询产品价格及折扣数据 3 汇集整理产品价格及折扣数据 4 制作产品价格表和销售折扣表 5 将产品价格表和销售折扣表传送给销售人员
可选事件流	1.6.1 更新产品价格及折扣数据 1 财务管理人员查询系统中原有的产品价格和折扣数据 2 将原有数据与当前数据对照，找出两个数据不相符合的产品 3 将原有数据更改为当前数据

	4 保存更改后的数据 1.6.2 增加新产品的价格及折扣数据 1 财务管理人员查询新加入的产品 2 查询新产品的价格及折扣数据 3 向系统中添加新产品的价格及折扣数据 4 保存新数据 1.6.3 删除已过时产品的价格及折扣数据 1 财务管理人员查询已过时的产品 2 在系统中删除已过时产品的价格及折扣数据 3 保存系统中现有产品的价格及折扣数据
特殊需求	无
前置条件	财务管理人员成功登陆财务管理子系统
后置条件	无
扩展点	产品价格表及销售折扣表已发送给销售人员
扩展用例	销售人员查收产品价格表及销售折扣表
2. 管理税率	
用例名称	管理税率
标识符	UC_Finance2
用例描述	财务管理人员对币种、汇率和税率进行管理
参与者	财务人员，销售人员
基本事件流	1 财务人员输入用户名和密码，登录财务管理子系统 2 查询币种、汇率和税率数据 3 汇集整理币种、汇率和税率数据 4 制作币种表、汇率表和税率表 5 将币种汇率税率数据传送给销售人员
可选事件流	2.6.1 更新币种汇率税率数据 1 财务管理人员查询系统中原有的币种、汇率和税率数据 2 将原有数据与当前数据对照，找出两个数据不相符合的产品 3 将原有数据更改为当前数据 4 保存更改后的数据
特殊需求	无
前置条件	财务管理人员成功登陆财务管理子系统
后置条件	无
扩展点	币种表、汇率表和税率表已发送给销售人员

扩展用例	销售人员查收币种表、汇率表和税率表
3. 处理客户汇款	
用例名称	处理客户汇款
标识符	UC_Finance3
用例描述	财务人员在确认客户汇款后改变客户付款状态
参与者	财务管理人员，客户，销售人员
基本事件流	3.5.1 处理预付款 1 客户汇出预付款 2 财务管理人员查询预付款是否到位 3 财务管理人员确认预付款金额是否满足要求 4 通知销售人员预付款已到 3.5.2 处理货款 1 客户汇出货款 2 财务管理人员查询货款是否到位 3 财务管理人员确认货款金额是否满足要求 4 通知销售人员货款已到
可选事件流	1. 若汇款的金额有误，则财务人员通知客户检查汇款金额
特殊需求	无
前置条件	客户汇款
后置条件	无
扩展点	1. 财务管理人员通知销售人员预付款已到 2. 财务管理人员通知销售人员货款已到
扩展用例	1. 销售人员填写预付款记录 2. 销售人员填写收款单
4. 处理发票	
用例名称	处理发票
标识符	UC_Finance4
用例描述	财务管理人员处理交易所需的发票
参与者	财务管理人员，客户，销售人员
基本事件流	1 销售人员向财务管理人员提交收款单并请求审核 2 财务管理人员审核收款单 3 财务管理人员制定发票 4 财务人员确认发票无误后发送给客户 5 客户得到发票

可选事件流	无
特殊需求	无
前置条件	销售人员向财务管理人员提交收款单
后置条件	无
扩展点	销售人员请求财务管理人员进行收款审核
扩展用例	销售人员填写收款单

5. 处理退货事宜

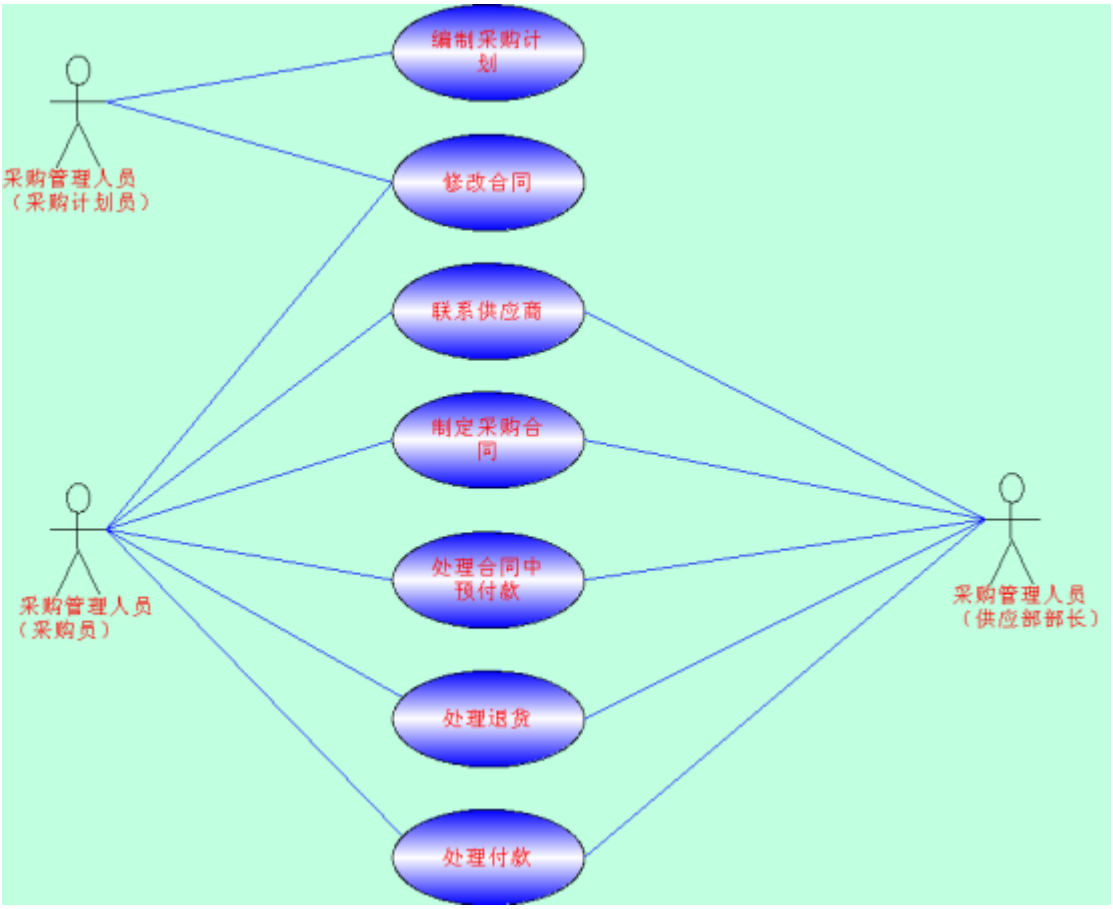
用例名称	处理退货事宜
标识符	UC_Finance5
用例描述	当客户退货时，财务人员查询退货单并退还已收货款
参与者	财务管理人员，客户，销售人员
基本事件流	1. 客户向销售人员发出退货要求 2. 销售人员制定退货单并报送销售计划员审批 2.1 若不同意该退货单，则通知销售员修改退货单 2.2 若同意该退货单，则将退货单交给成品库保管员并通知其退货 2.3 成品库保管员将退货入库并通知销售人员已收到退货 3. 销售人员向财务人员发送退款单并通知其退款 4. 财务人员审核退款单并退款 5. 客户得到退款并确认退款单
可选事件流	1. 若成品库保管员未收到退货，则不予退款 2. 若退款单未通过审核，则返回给上一步相关人员重新填写
特殊需求	无
前置条件	客户请求退款
后置条件	客户收到退款
扩展点	1 销售人员请求财务管理人员审核退款单 2 财务人员向客户提交退款单
扩展用例	1 销售人员制定退款单 2 客户领取退款

6. 备案各种票据

用例名称	备案各种票据
标识符	UC_Finance6
用例描述	财务人员将各种单据如销售订单、出入库单等备案以供将来查询
参与者	财务人员

基本事件流	1 财务管理人员登陆财务管理子系统 2 财务管理人员汇集销售订单、出入库单和发货单存根等各种单据 3 对各种单据进行审核并确认无误 4 对各种单据进行备案
可选事件流	无
特殊需求	无
前置条件	财务人员登陆系统
后置条件	无
扩展点	无
扩展用例	无

2.采购管理人员相关用例



编制采购计划

用例名称	编制采购计划
标识符	UC_Purchase1
用例描述	采购计划员制定公司的采购计划
参与者	采购管理人员（采购计划员）

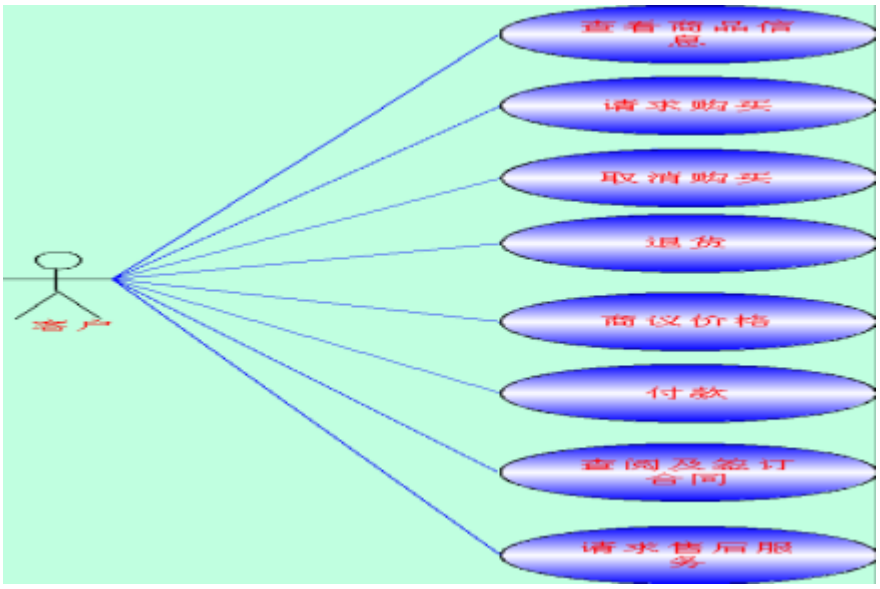
基本事件流	1. 查询物资需求计划 2. 制定采购需求计划 3. 申请批复采购需求计划 4. 编制采购计划
可选事件流	无
特殊需求	无
前置条件	企业各部门向采购管理人员呈报物资需求计划表
后置条件	无
扩展点	无
扩展用例	无
联系供应商	
用例名称	联系供应商
标识符	UC_Purchase2
用例描述	采购计划员制定公司的采购计划
参与者	采购管理人员（采购员，供应部部长）
基本事件流	1. 查询企业的采购计划 2. 向供应商询问相关产品价格 3. 判断该供应商是否为新供应商 3.1.1 若是新供应商，则建立该供应商管理档案 3.1.2 若是老供应商，则比较供应商的报价单比价和供应商的信用度 4. 选定供应商 5. 向供应部部长送审选定供应商 6. 供应部部长审批选定供应商
可选事件流	无
特殊需求	无
前置条件	无
后置条件	无
扩展点	无
扩展用例	无
制定采购合同	
用例名称	制定采购合同
标识符	UC_Purchase3
用例描述	采购人员制定所采购物资的采购合同

参与者	采购管理人员（采购员，供应部部长）
基本事件流	1. 供应部部长通知采购员签订采购合同 2. 采购员制定采购合同 3. 采购员将采购合同送交供应商审查 4. 供应商返回采购合同 5. 采购员将采购合同送交供应部部长审核 6. 供应部部长审批采购合同
可选事件流	无
特殊需求	无
前置条件	供应部部长同意采购合同
后置条件	无
扩展点	无
扩展用例	无
处理合同中的预付款	
用例名称	处理合同中的预付款
标识符	UC_Purchase4
用例描述	采购管理人员处理采购合同中的预付款项
参与者	采购管理人员（采购员，供应部部长）
基本事件流	1. 供应部部长批复采购合同 2. 采购员确认采购合同中是否有预付款项 3. 若有预付款，则申请预付款单 4. 采购员将预付款单送交供应部部长审批 5. 供应部部长审批预付款单
可选事件流	无
特殊需求	无
前置条件	合同中有预付款项
后置条件	无
扩展点	无
扩展用例	无
处理退货	
用例名称	处理退货
标识符	UC_Purchase5
用例描述	采购管理人员处理退货事宜
参与者	采购管理人员（采购员，供应部部长）

基本事件流	1. 物资质量检验人员通知采购员购件不合格 2. 采购员制定退货单 3. 采购员向供应部部长送审退货单 4. 供应部部长批复退货单 5. 采购员确认是否已经付款 1. 若已付款则要求供应商退货退款 2. 若未付款则要求供应商退货
可选事件流	无
特殊需求	无
前置条件	所采购物资质量不合要求
后置条件	无
扩展点	无
扩展用例	无
处理付款	
用例名称	处理付款
标识符	UC_Purchase6
用例描述	采购管理人员处理采购活动中的付款
参与者	采购管理人员（采购员，供应部部长）
基本事件流	1. 库存管理人员通知所购物资已入库 2. 采购员向供应商所要发票 3. 供应商向采购员提供发票 4. 采购员制定采购付款单 5. 采购员向供应部部长送审采购收款单 6. 供应部部长审批采购收款单
可选事件流	无
特殊需求	无
前置条件	所有采购物资已入库
后置条件	无
扩展点	无
扩展用例	无
修改合同	
用例名称	修改合同
标识符	UC_Purchase7
用例描述	采购计划人员在修改采购计划后通知采购员相应地修改采购合同

参与者	采购管理人员（采购员，供应部部长）
基本事件流	1. 采购计划人员通知采购员采购计划已修改 2. 采购员核对采购合同 3. 采购员确定采购合同是否应该中止 1. 若采购合同应该中止，则中止相应采购合同 2. 若采购合同无需中止，则修改合同以适应采购计划的修改
可选事件流	无
特殊需求	无
前置条件	采购计划已经修改
后置条件	无
扩展点	无
扩展用例	无

3.客户相关用例



查看商品信息

用例名称	查看商品信息
标识符	UC_Customer1
用例描述	这个用例描述了客户查看商品信息的流程
参与者	客户
基本事件流	客户从销售人员或是门户网站查看商品信息
可选事件流	无
特殊需求	无

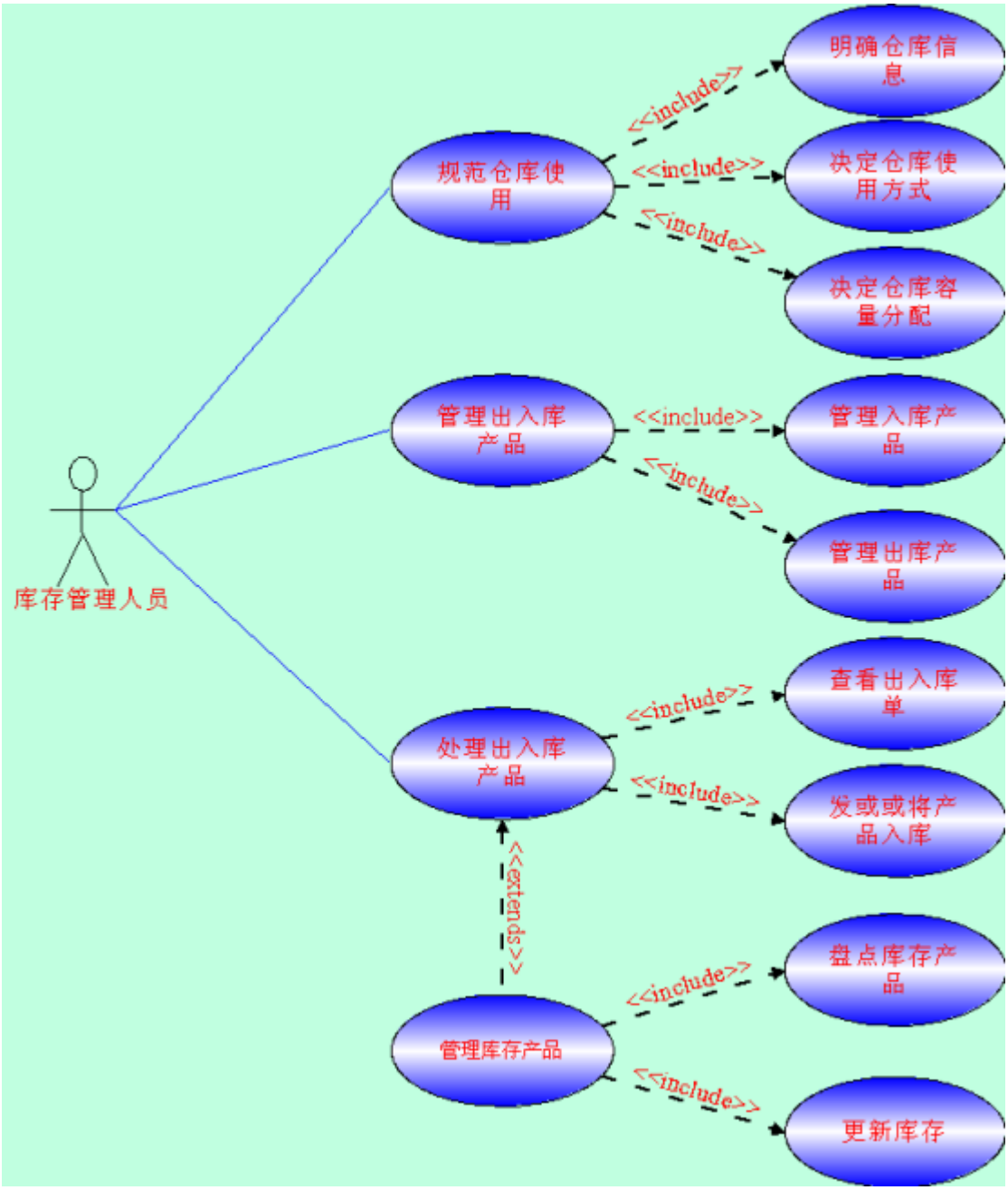
前置条件	客户有购买可能
后置条件	提出购买商品
扩展点	无
扩展用例	无
商品价格面议	
用例名称	商品价格面议
标识符	UC_Customer2
用例描述	这个用例描述了客户商品价格商议的流程
参与者	客户
基本事件流	1. 客户查看商品信息 2. 客户向销售人员问价 3. 客户得到报价单 4. 客户与销售人员商议价格
可选事件流	1. 客户得不到报价 2. 无法确定商品价格
特殊需求	无
前置条件	查看商品信息
后置条件	提出购买商品
扩展点	无
扩展用例	无
请求购买商品	
用例名称	请求购买商品
标识符	UC_Customer3
用例描述	这个用例描述了客户购买商品的流程
参与者	客户
基本事件流	1. 客户得到商品报价 2. 客户向销售人员提出购买商品请求
可选事件流	1. 销售人员否定购买商品请求
特殊需求	无
前置条件	商品问价
后置条件	业务机会出现，销售人员创建销售订单请求
扩展点	无
扩展用例	无
商议和签订合同	

用例名称	商议和签订合同
标识符	UC_Customer4
用例描述	这个用例描述了客户合同处理的流程
参与者	客户
基本事件流	1. 客户得到销售人员发来的合同 2. 客户同意合同内容 3. 客户签订合同
可选事件流	客户否定合同，要求修改合同内容
特殊需求	无
前置条件	商品购买事宜确定
后置条件	销售订单创建成功，客户付款，库存部门准备发货
扩展点	无
扩展用例	无
取消购买商品	
用例名称	取消购买商品
标识符	UC_Customer5
用例描述	这个用例描述了客户取消购买商品的流程
参与者	客户
基本事件流	1. 要求客户签订合同 2. 客户不想购买商品，决定取消购买行为 3. 客户不签订合同，取消购买
可选事件流	销售管理人员不同意取消合同
特殊需求	无
前置条件	客户被要求签订合同
后置条件	业务机会取消，购买行为无效
扩展点	无
扩展用例	无
客户退货	
用例名称	客户退货
标识符	UC_Customer6
用例描述	这个用例描述了客户退货的流程
参与者	客户
基本事件流	1. 客户商品已经到达客户目的地 2. 客户不需要商品了，要求退货

	3. 客户将已购商品退还商家 4. 客户收到退货款项
可选事件流	1. 销售管理人员不同意退货，可能发生纠纷 2. 财务人员不同意退款
特殊需求	无
前置条件	客户已经得到商品
后置条件	客户收到退款，商品退还入库
扩展点	无
扩展用例	无
客户为购买商品付款	
用例名称	客户为购买商品付款
标识符	UC_Customer7
用例描述	这个用例描述了客户付款的流程
参与者	客户
基本事件流	1. 客户签订合同 2. 客户使用不同支付方式向财务部门支付商品款项 3. 客户收到发票和收据
可选事件流	支付过程出现问题，付款未能到达财务部门
特殊需求	无
前置条件	客户签订合同，按照要求需要付款
后置条件	客户收到相关票据，收据和发票，货物发出
扩展点	无
扩展用例	无
客户要求售后服务	
用例名称	客户要求售后服务
标识符	UC_Customer8
用例描述	这个用例描述了客户要求售后服务的流程
参与者	客户
基本事件流	1. 客户商品出现服务相关问题 2. 客户使用不同方式向售后部门提出售后服务要求 3. 售后部门负责处理服务请求
可选事件流	售后服务部门不受理服务请求
特殊需求	无
前置条件	客户拥有商品出现问题

后置条件	售后服务部门提供服务
扩展点	无
扩展用例	无

4.库存管理人员相关用例



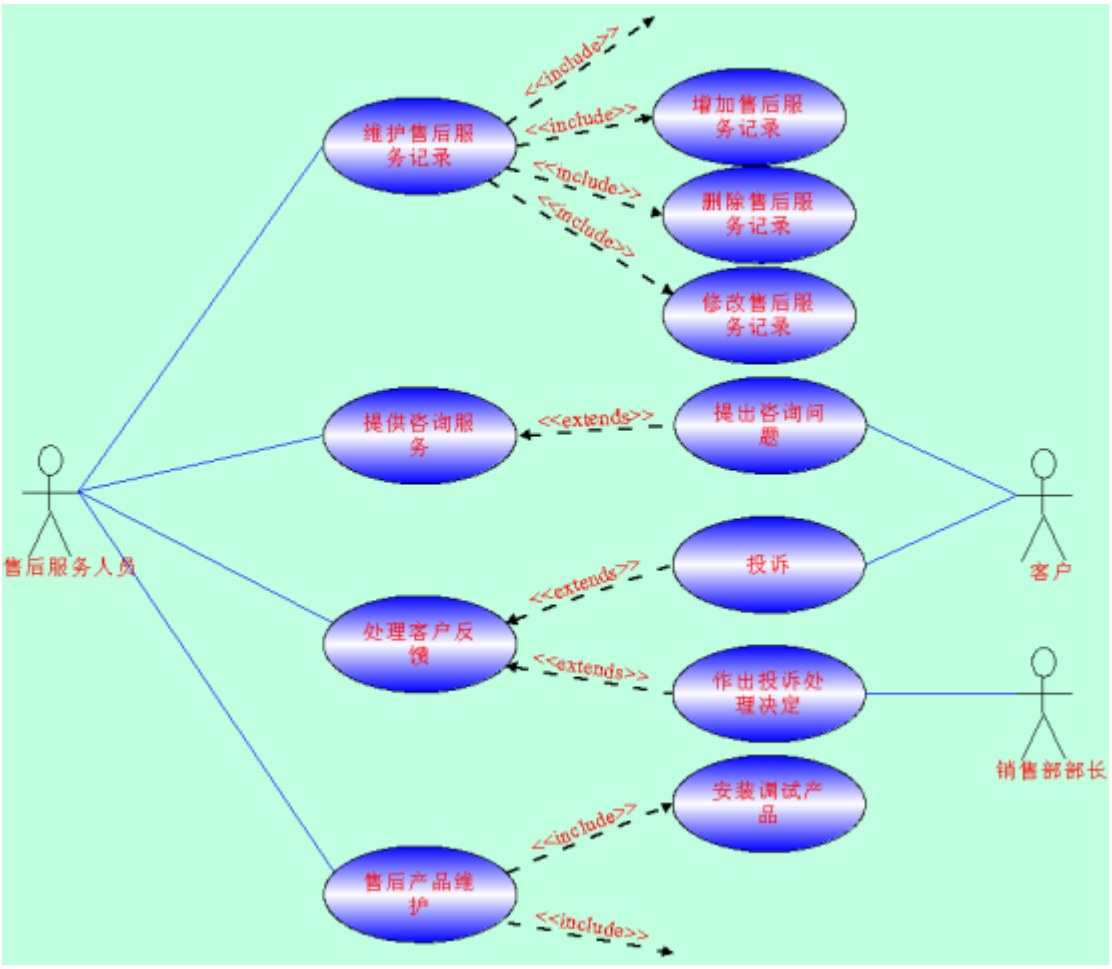
规划仓库使用

用例名称	规划仓库使用
标识符	UC_Warehouse1
用例描述	这个用例描述了库存管理人员规划管理仓库的流程

参与者	库存管理人员
基本事件流	1. 库存管理人员明确仓库相关信息 2. 库存管理人员决定仓库使用方式 3. 库存管理人员决定仓库容量分配
可选事件流	仓库不够使用时，设法扩展仓库
特殊需求	无
前置条件	库存管理人员了解仓库信息
后置条件	按照库存管理人员分配使用仓库
扩展点	无
扩展用例	无
管理出入库单商品	
用例名称	管理出入库单商品
标识符	UC_Warehouse2
用例描述	这个用例描述了库存管理人员处理出入库单的流程
参与者	库存管理人员
基本事件流	1. 库存管理人员查看销售管理人员发来的销售订单 2. 库存管理人员根据销售订单情况填写出入库单
可选事件流	销售订单没有到达
特殊需求	无
前置条件	有销售订单到来，需要出入库处理
后置条件	商品出入库处理
扩展点	无
扩展用例	无
出入库处理	
用例名称	出入库处理
标识符	UC_Warehouse3
用例描述	这个用例描述了库存管理人员处理出入库工作的流程
参与者	库存管理人员
基本事件流	1. 库存管理人员查看出入库单 2. 库存管理人员发货或是将商品入库 3. 库存管理人员更新库存量记录，UC_warehouse4
可选事件流	1. 库存不足以满足出库单 2. 仓库不能存放入库单中的商品量
特殊需求	无

前置条件	已经有出入库单，需要出入库商品
后置条件	物流人员运输商品
扩展点	无
扩展用例	无
管理库存量	
用例名称	管理库存量
标识符	UC_Warehouse4
用例描述	这个用例描述了库存管理人员管理库存量的流程
参与者	库存管理人员
基本事件流	1. 处理出入库商品 2. 库存管理人员记录库存量变更 3. 库存管理人员盘点仓库库存量
可选事件流	无
特殊需求	无
前置条件	无
后置条件	库存量记录变更
扩展点	无
扩展用例	无

5.售后服务人员相关用例



维护售后服务记录

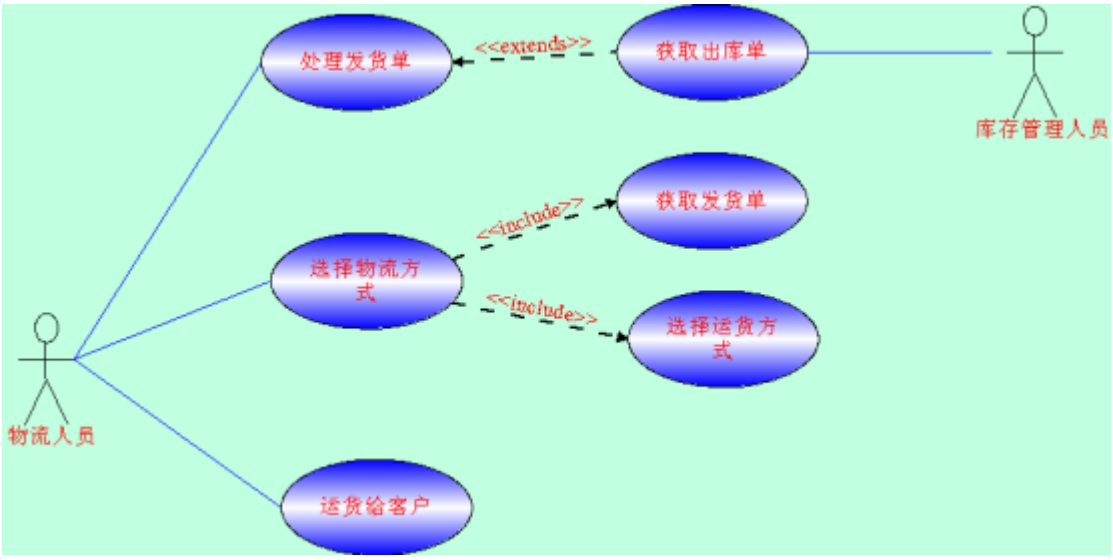
用例名称	维护售后服务记录
标识符	UC_Sales1
用例描述	售后服务人员对售后服务纪录进行查询、修改、增加或删除等操作
参与者	售后服务人员
基本事件流	1. 售后服务人员登陆系统并进入售后服务记录子系统 2. 查询售后服务纪录 3. 修改有误的售后服务纪录 3.1 查找错误的售后服务纪录 3.2 确定错误之处并加以修改 3.3 保存修改后的纪录 3.4 更新售后服务纪录 4. 增加新的售后服务纪录 4.1 在原有的售后服务纪录中增加新的服务纪录

	4.2 保存新增的服务纪录 4.3 更新售后服务纪录 5. 删除无效的售后服务纪录 5.1 查找无效的售后服务纪录 5.2 删除无效的售后服务纪录 5.3 更新售后服务纪录
可选事件流	1. 若登陆系统失败，则需检查账号和密码并重新登陆 2. 若查询、修改、增加和删除失败，则需退出售后服务记录子系统并重新进入和进行相关操作
特殊需求	无
前置条件	售后服务人员登陆系统
后置条件	售后服务纪录得到正确维护
扩展点	无
扩展用例	无
提供咨询服务	
用例名称	提供咨询服务
标识符	UC_Sales2
用例描述	售后服务人员为客户提供咨询
参与者	售后服务人员，客户
基本事件流	1. 客户提出要咨询的问题 2. 售后服务人员针对该问题确定相应的答案 3. 售后服务人员将答案反馈给客户
可选事件流	若售后服务人员无法解答客户咨询的问题，则应向上级部门申报并将上级部门的处理意见反馈给客户
特殊需求	无
前置条件	售后服务人员登陆系统
后置条件	客户获得所咨询问题的答案
扩展点	无
扩展用例	无
售后产品维护	
用例名称	售后产品维护
标识符	UC_Sales3
用例描述	售后服务人员为客户提供产品的安装调试和维修等服务
参与者	售后服务人员，客户，销售人员

基本事件流	1. 用户购买相关产品 2. 售后服务人员为客户安装调试新产品 2.1 销售人员将销售单传给售后服务人员并通知其安装 2.2 售后服务人员中的调度人员填写安装调试单并将安装调试任务分配给维修人员 2.3 维修人员安装调试产品，填写安装调试记录单并将其交给调度人员 2.4 调度人员审核安装调试单，填写安装调试纪录 3. 当产品出故障时，售后服务人员为客户提供维修服务 3.1 接待员接待客户来电来函，登记处理单并报送调度人员 3.2 调度人员填写产品维修单，并将维修任务分配给维修人员 3.3 维修人员维修产品并现场填写产品维修单纪录，然后将其报送给调度人员 3.4 调度人员审核产品维修单纪录，填写产品维修单
可选事件流	1. 若产品安装、调试或维修过程中出现维修人员不能解决的问题，则由调度人员指派新的维修人员 2. 若安装调试或维修纪录填写不正确、不规范，则调度人员应向相关负责人征询并追加相应细节
特殊需求	无
前置条件	客户购买相关产品
后置条件	售后服务人员保证客户购买的产品正常使用
扩展点	无
扩展用例	无
处理客户反馈	
用例名称	处理客户反馈
标识符	UC_Sales4
用例描述	售后服务人员对客户的投诉进行处理
参与者	售后服务人员，客户，销售部部长
基本事件流	1. 客户提出投诉 2. 接待人员填写客户投诉处理单并将之上报给销售部部长 3. 销售部部长填写客户投诉处理意见 4. 如果销售部部长做出退货决定，则会通知调度人员制定退货单并执行退货的相关操作 5. 如果销售部部长做出维修决定，则会通知调度人员填写产品维修

	单，对客户的产品进行维修 6. 如果销售部部长做出更换或赔偿决定，则安排更换或赔偿并让接待人员转达客户该处理结果
可选事件流	无
特殊需求	无
前置条件	客户提出投诉
后置条件	客户的投诉得到妥善处理
扩展点	无
扩展用例	无

6.物流人员相关用例

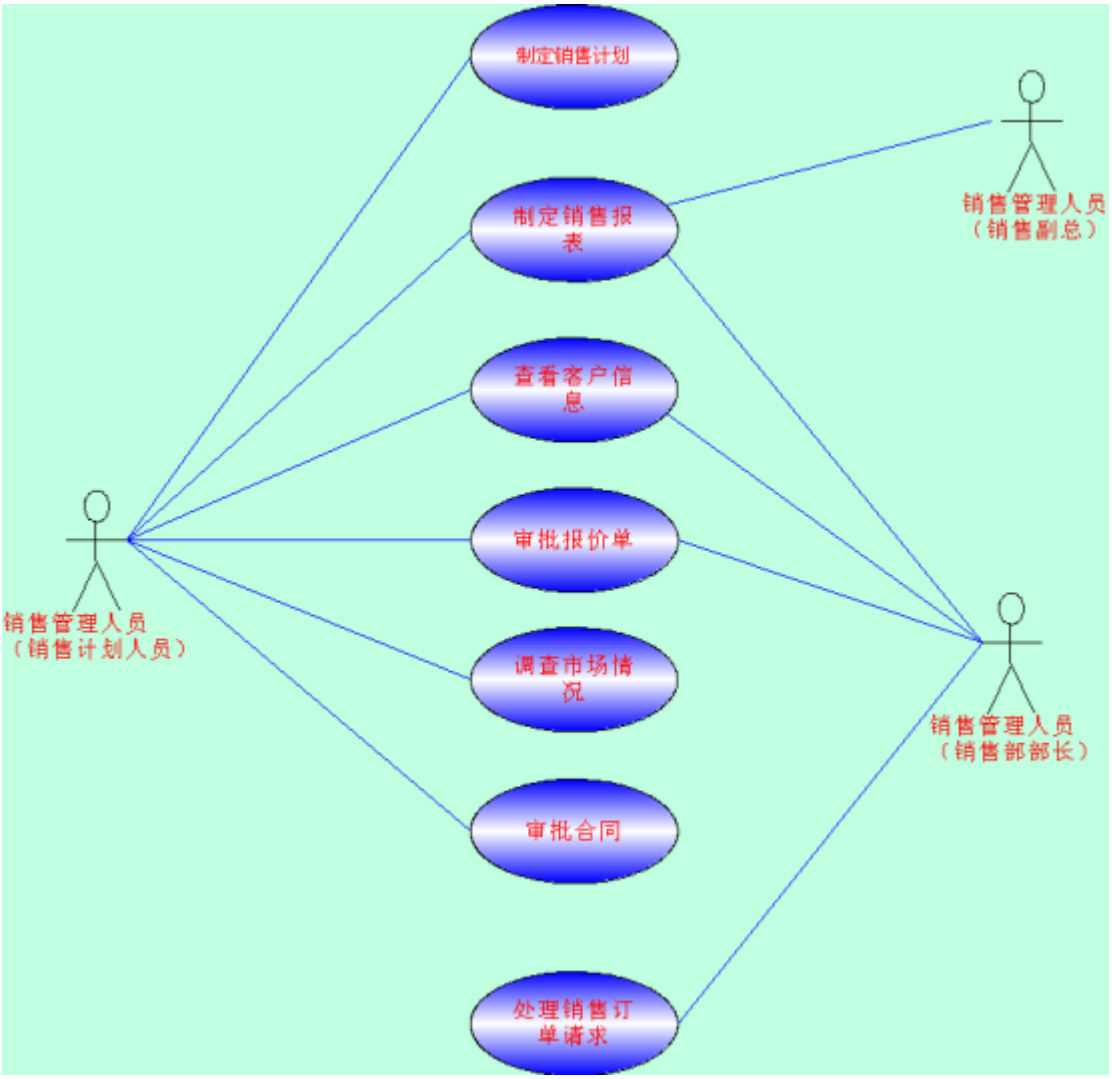


处理发货单

用例名称	处理发货单
标识符	UC_Delivery1
用例描述	这个用例描述了物流人员处理发货单的流程
参与者	物流人员
基本事件流	1. 物流人员从库存管理人员处得到出库单 2. 根据出库单信息填写发货单内容单，对客户的产品进行维修
可选事件流	出库单信息和实际货物信息不合
特殊需求	无
前置条件	从库存管理人员处得到出库单
后置条件	从库存管理人员处得到出库单
扩展点	无

扩展用例	无
选择物流方式	
用例名称	选择物流方式
标识符	UC_Delivery2
用例描述	这个用例描述了物流人员选择物流方式的流程
参与者	物流人员
基本事件流	1. 物流人员收到出库单，填写发货单 2. 根据发货地点和要求选择物流方式
可选事件流	选择的物流方式目前不可行
特殊需求	无
前置条件	确定了应有的发货信息
后置条件	运货给客户
扩展点	无
扩展用例	无
运货给客户	
用例名称	运货给客户
标识符	UC_Delivery3
用例描述	这个用例描述了物流人员运货的流程
参与者	物流人员
基本事件流	1. 库存处取运输商品 2. UC_delivery1，物流人员填写发货单 3. UC_delivery2，物流人员选择物流方式 4. 运货到目的地
可选事件流	1. 没有足够库存，目前无法发货 2. 运输方式不可行
特殊需求	无
前置条件	收到出库单，去库存人员处取货
后置条件	运输商品到达目的地
扩展点	无
扩展用例	无

7.销售管理人员相关用例



审批报价单

用例名称	审批报价单
标识符	UC_SalesManager1
用例描述	销售管理人员对销售人员提交的报价单进行审批
参与者	销售管理人员（销售部部长，销售副总）
基本事件流	1. 销售人员报审报价单 2. 销售部部长查收新的报价单 3. 审阅报价单，判断该报价单是否为重要报价单 1. 若不是重点报价单，则对报价单做出批复，并将其返回给销售人员 2. 若是重点报价单，则向销售副总报批报价单，销售副总批复报价单并将其返回给销售人员
可选事件流	1. 销售部长不同意报价单，将其返回给销售人员加以修改并重新报

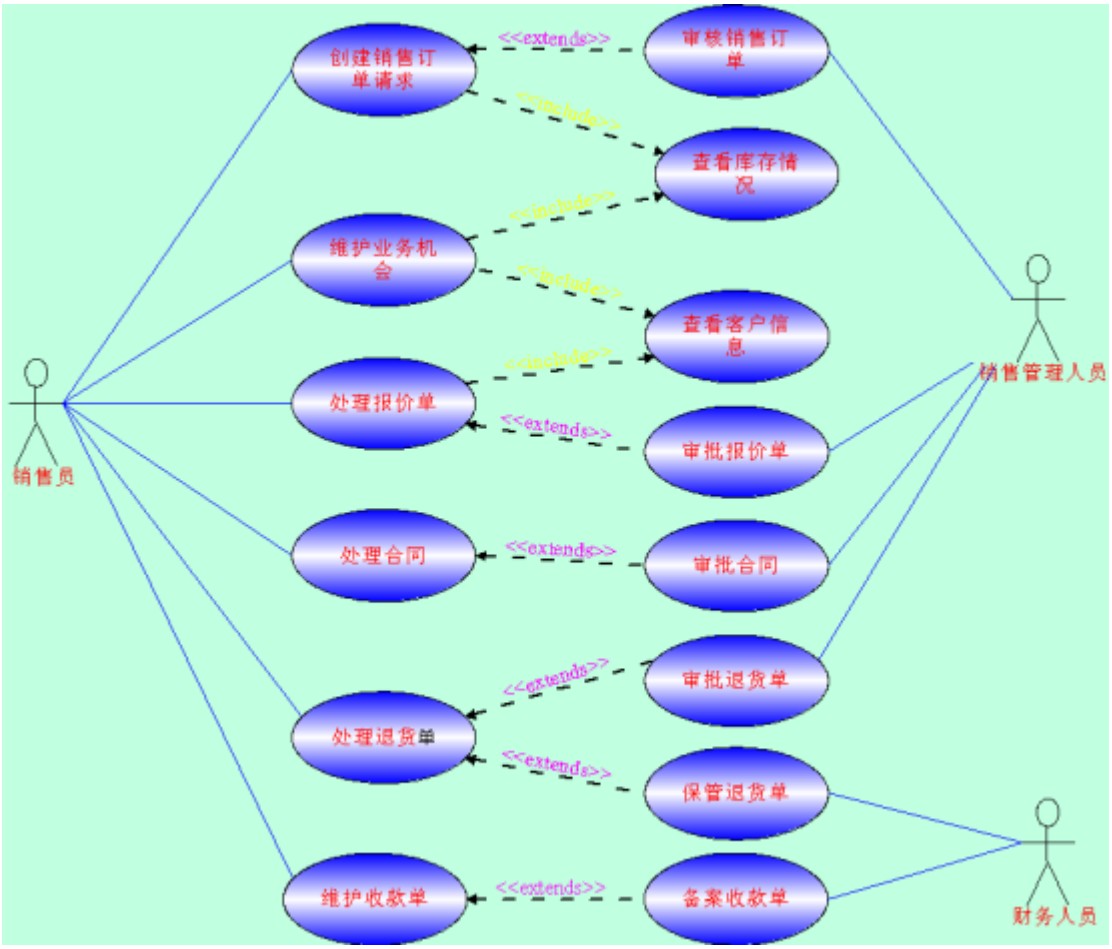
	审 2. 销售副总不同意重要报价单，将其返回给销售人员加以修改并重新报审
特殊需求	无
前置条件	销售人员填制好报价单
后置条件	运输商品到达目的地
扩展点	无
扩展用例	无
制定销售计划	
用例名称	制定销售计划
标识符	UC_SalesManager2
用例描述	销售计划员根据合同及收到的预付款制定相应的销售计划
参与者	销售管理人员（销售计划员）
基本事件流	1. 销售人员通知销售计划员收到预付款 2. 销售计划员查收并核对预付款 3. 根据收到的预付款制定相应的销售计划
可选事件流	无
特殊需求	无
前置条件	销售人员收到客户的预付款
后置条件	无
扩展点	无
扩展用例	无
制定销售报表	
用例名称	制定销售报表
标识符	UC_SalesManager3
用例描述	销售计划员根据合同及收到的预付款制定相应的销售计划
参与者	销售管理人员（销售计划员，销售部部长，销售副总）
基本事件流	1. 销售计划人员收集统计公司的销售情况及相关数据 2. 销售计划人员将收集的数据加以整理并做出销售报表（包括销售日报、销售月报和地区销售统计表等） 3. 销售计划人员将销售报表送给销售部部长审阅 4. 销售部部长审阅通过后再将销售报表交给销售副总审阅 5. 销售计划人员保存通过审阅的销售报表
可选事件流	无

特殊需求	无
前置条件	销售计划人员得到公司产品的销售资料及数据
后置条件	无
扩展点	无
扩展用例	无
查看客户信息	
用例名称	查看客户信息
标识符	UC_SalesManager4
用例描述	销售管理人员在需要的时候查看客户的相关信息
参与者	销售管理人员
基本事件流	1. 销售管理人员登陆系统 2. 销售管理人员输入要查询的信息如客户名称、帐号等 3. 系统返回查询结果
可选事件流	无
特殊需求	无
前置条件	销售管理人员需要得知客户的相关信息
后置条件	无
扩展点	无
扩展用例	无
调查市场状况	
用例名称	调查市场状况
标识符	UC_SalesManager5
用例描述	销售管理人员调查市场对相关产品的供需情况
参与者	销售管理人员（销售计划员）
基本事件流	1. 销售管理人员制定相关产品的市场调查计划 2. 销售管理人员安排实施调查的相关人员 3. 市场调查人员将调查结果反馈给销售管理人员 4. 销售管理人员对调查结果进行整理和分析 5. 销售管理人员将结果报送有关部门和领导
可选事件流	无
特殊需求	无
前置条件	公司决定对市场状况进行调查
后置条件	无
扩展点	无

扩展用例	无
审批合同	
用例名称	审批合同
标识符	UC_SalesManager6
用例描述	销售管理人员审批销售人员报送的合同
参与者	销售管理人员（销售计划员）
基本事件流	1. 销售人员向销售管理人员报审合同 2. 销售管理人员审查合同细节 1. 若同意合同内容，则将合同审核意见返回给销售人员 2. 若不同意合同内容，则将合同返回给销售人员并要求其加以修改 3. 销售人员得到批复后的合同
可选事件流	1. 销售人员向销售管理人员报批合同修改单 2. 销售管理人员审核合同修改单并将其返回给销售人员 3. 销售人员得到批复后的合同修改单并对合同加以修改
特殊需求	无
前置条件	销售人员填写好合同并提交给销售管理人员
后置条件	无
扩展点	无
扩展用例	无
处理销售订单请求	
用例名称	处理销售订单请求
标识符	UC_SalesManager7
用例描述	销售管理人员处理销售人员提交的销售订单
参与者	销售管理人员（销售计划员）
基本事件流	1. 销售人员向销售管理人员请求审批销售订单 2. 销售管理人员审核销售订单的细节 3. 销售管理人员给出审批意见并将销售订单返回给销售人 4. 销售人员拿到审批后的销售订单
可选事件流	无
特殊需求	无
前置条件	销售人员提交销售订单
后置条件	无
扩展点	无

扩展用例	无

8.销售人员相关用例



服务模型与设计

引言

什么是服务？很难给出一个精确的定义。在 IBM 的服务科学，工程与管理的背景之下，面向服务的架构关注的是企业业务与信息技术之间的协调性和一致性。（The alignment of organizational IT with its Business Strategy, Structure, and Process）。一般地讲，服务是一种新型的应用程序， 具有自包含、自描述以及模块化的特点，可以通过互联网发布、查找和调用。以下是对服务的几种定义：

“服务是基于开放标准的可通过互联网访问的业务逻辑。”

“Web Service is a piece of business logic accessible via the Internet， using open standard.” -- Microsoft

“服务是在互联网上、通过标准协议提供的、独立封装的、松散耦合的、按合同运作的软件功能体。”

“Encapsulated， loosely coupled， contracted software function， offered via standard protocols over the web.” -- DestiCorp

“服务是松散耦合的软件组件，其在标准的互联网技术的支撑下互相通信、交互，形成动态的网络关系。”

“Loosely coupled software components that interact with one another dynamically via standard Internet technologies.” -- Garnter

“Web 服务是可以通过统一资源描述符定义的软件应用，其接口和绑定实现可用 XML 文件格式定义、描述和搜索，能够基于互联网的标准协议与其他软件应用直接进行 XML 格式的消息传递。”

“A software application identified by a URI， whose interfaces and bindings are capable of being defined， described， and discovered by XML artifacts and supports direct interactions with other software applications using XML-based messages via Internet-base d protocols.” --World Wide Web Consortium(W3C)

不同研究领域对 Web 服务的关注点是不同的：

——在网络研究中，Web 服务的性质由其带宽，可用性，故障率及相关的属性；

——在通信领域研究中，Web 服务相当于某种电话功能属性，如来电显示、呼叫转移以及基本的接通服务，如宽待与窄带；

——在系统工程研究中，Web 服务特指收费、保存以及其他的操作功能，这类功能也通常被包装成运行支撑系统；

——在网络应用研究中，Web 服务被定义为 Web 页面，尤其是那些包含表格和程序接口的交互式页面；

——在无线网络研究中，Web 服务包括短信等通用服务。

在上述的服务定义中，Web 服务的共同特征就在于它是可被提供或者利用的某种功能。我们综合上述定义，可以将 Web 服务定义为：Web 服务是可通过互联网定购的某种能力（包括功能和质量）。

通常地，业务服务应该与服务的质量分别处理，比如说安全性，可靠性，或者交互性的方面要求被认为是质量相关的，非功能性的。同时业务服务还要独立于传输实现技术，就是各个服务之间互相联系的底层传输机制也可以不同于其他节点。服务的重要之处在于它必须要实施拟定的业务流程，比如说一个贷款应用服务与邮件审批服务、信用评级、与支付服务之间是互相联系的，里面当然可能包含一些人工服务。所有这些服务从敏捷 SOA 方法的角度来看它不仅包含技术性的服务，同时必须是信息驱动的，也就是说它们不是进行参数传递，而是进行文档的传输，应答与否也是可选的。我们可以把新的业务程序通过对旧有的流程重组之后再融入到现在有关的程序当中。也就是可以把一个业务部件用一个新的等价部件替换。也可将新的业务流程融入旧的业务流程中去，领域之间也要有一定的独立性。所有的这些服务都不知道谁是客户，谁是顾客，这些服务的开发应该是相对独立的。

服务定义涉及粒度的问题，服务的粒度的划分是核心问题之一。服务请求者为了完成一项业务操作而对服务提供者进行的服务调用次数决定了服务的粒度。当需要进行多个服务调用时，服务提供者需要实现细粒度的服务。如果只需进行少量的服务调用，那么服务提供者就只需要实现粗粒度的服务。虽然细粒度的接口为请求者应用程序提供了更多的灵活性，它同样也意味着交互的模式可能随着不同的服务请求者而不同，从而使对于服务提供者的支持更加困难和昂贵。粗粒度接口保证服务请求者以相对一致的方式使用服务。

在实际工作中的服务相关工作涉及到几个方面：服务发现、服务列表和服务实现。对于定义服务来说这是一套完整的流程工作。下面我们来依次分析。

1. 服务发现

服务发现是根据业务相关活动的需求选择可能的候选服务，这些候选服务将通过一定的规则进行筛选，筛选的结果导致正式的服务关系。对 SOA 来说，服务是一个基本组成单元，是 SOA 的基本概念，而服务发现是服务相关工作的基础设施的重要组成部分。服务发现是后续服务相关活动的基础，我们通过以下三

步来实现服务的发现过程:

1.1 服务发现第一步：业务流程分解方法

业务流程分析方法是通过对于主要业务流程的分解，把所有可能涉及的流程活动罗列出来，形成候选服务列表。业务流程的分析我们在业务模型文档中详细加以说明，这里只是强调如何利用业务流程的分析结果来发现候选服务。业务流程分解法是很直观的方法，对于所有在流程中出现的所有业务活动，我们都作为候选服务来考虑。

针对每一个业务流程中出现的候选服务，我们粗略的对其进行分类，分类的标准参照的是 SOA 实现中的参考架构，类型包括可重用的应用服务、流程服务和人工服务。这种分类方法比较简单。

可以按照 SOA 的参考架构来对服务进行分类，分为业务逻辑服务、控制服务、连接服务、业务创新和优化服务、开发服务、IT 服务管理，这个分类也将成为我们在 SOA 的 ESB 实现中会考虑的分类方式。但是在这里我们只是简单的分类。

下面是对每一个分解了的业务流程进行候选服务选择，大部分的流程活动会成为候选服务，也有一些很少调用的服务会被筛选掉。

①创建业务机会流程

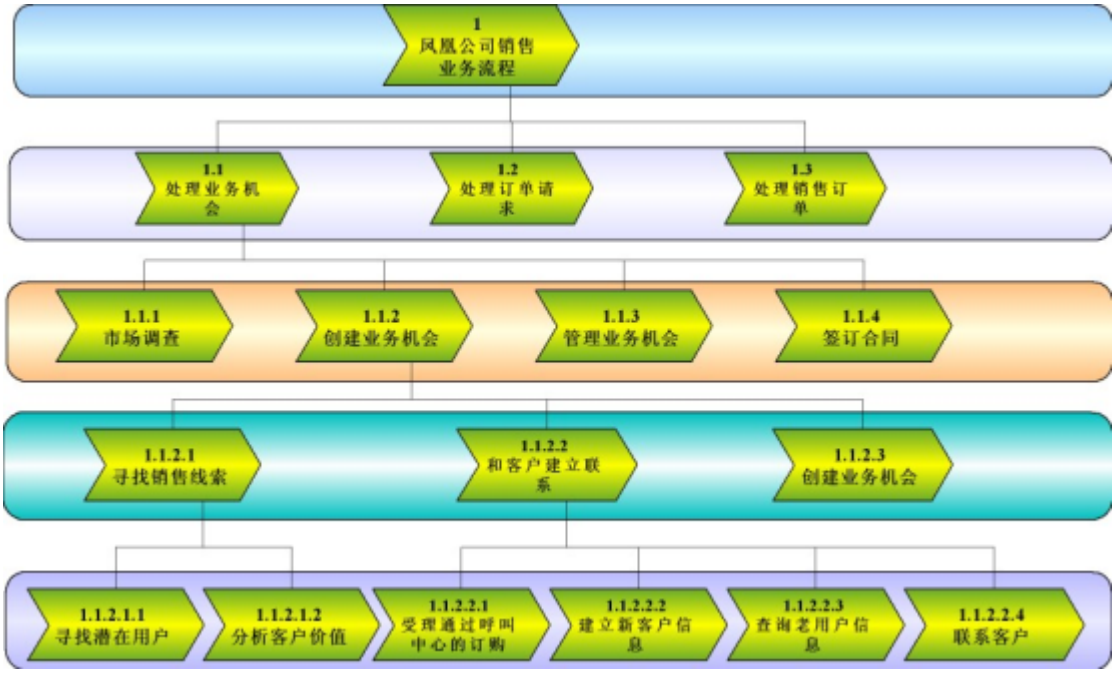


图 1 创建业务机会流程

我们把图中的标号和服务对应起来，这个图中其中包含的流程和分类如下:

1.1 创建业务机会：属于流程服务

1.2 处理订单请求：属于流程服务

1.3 处理销售订单：属于流程服务

1.1.2.1 寻找商机/销售线索：属于人工服务

1.1.2.2.4 联系客户/呼叫客户：属于应用服务

1.1.2.3 创建业务机会记录：属于应用服务

1.1.2.2.3 查看客户信息：属于应用服务

1.1.2.2.2 建立客户信息：属于应用服务

②管理销售机会流程

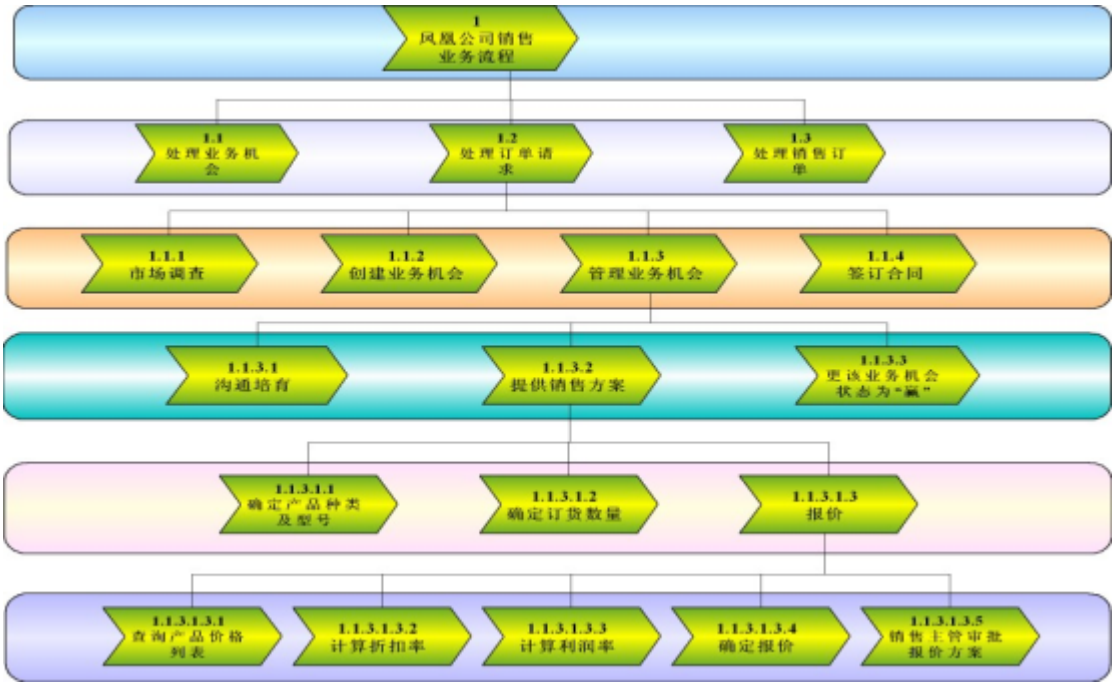


图 2 管理销售机会流程

1.1.3.3 更改业务机会状态：属于应用服务

1.1.3.1.3.4 创建报价单：属于应用服务

1.1.3.1.3 报价：属于流程服务

1.1.3.1.3.1 查看商品价格列表：属于应用服务

1.1.3.1.3.2 计算折扣率：属于功能服务，应用服务

1.1.3.1.3.3 计算利润率：属于功能服务，应用服务

1.1.3.1.3.5 审核报价方案：属于人工服务

③签订合同流程

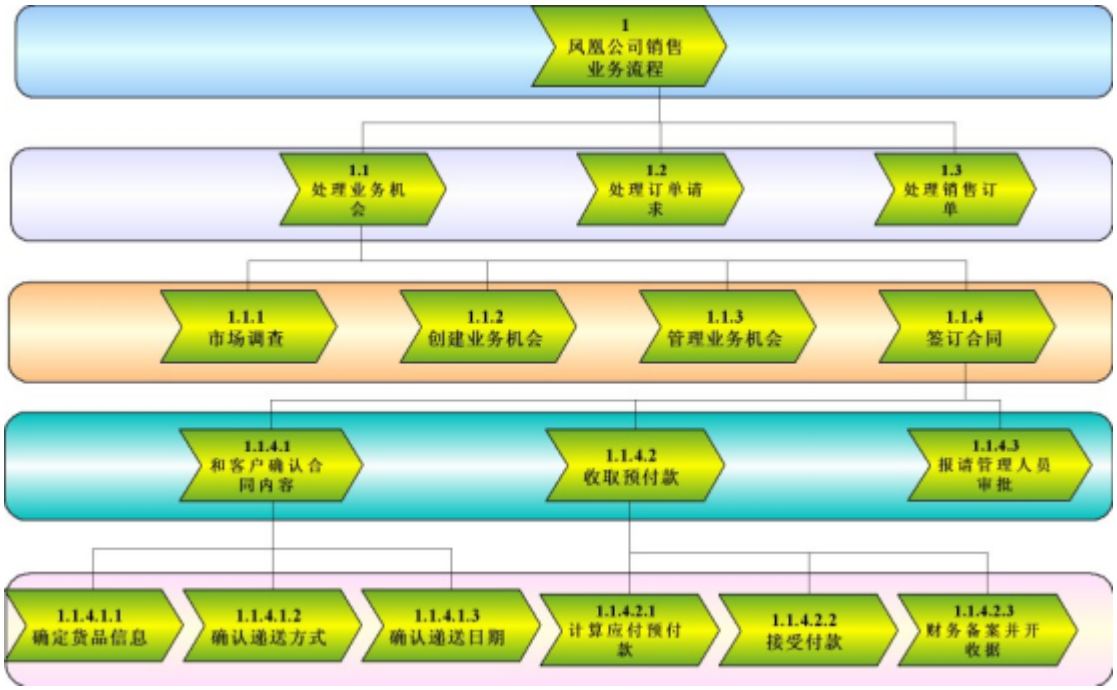


图 3 签订合同流程

1.1.4 签订合同：属于人工服务

1.1.4.2.3 收钱帐目变更：属于应用服务

1.1.4.2.3 财务备案并开收据：属于应用服务

④验证订单请求

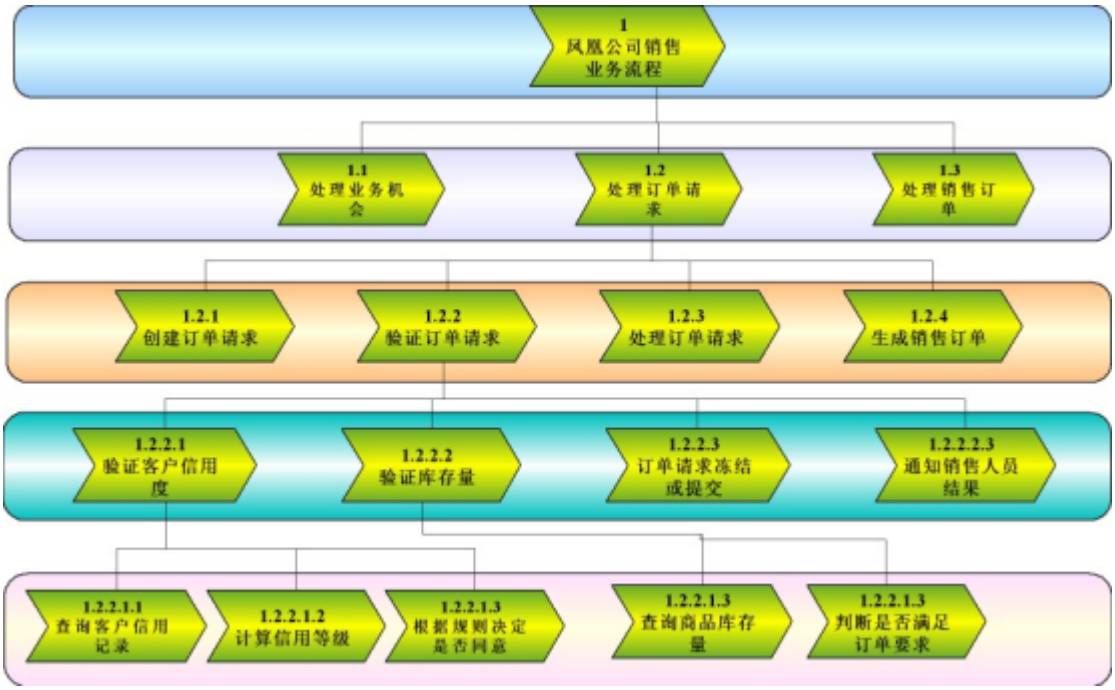


图 4 验证订单请求流程

1.2.2.1 验证客户信用度：属于应用服务

1.2.2.2 验证库存量：属于应用服务

1.2.2.1.1 查看客户信用记录：属于应用服务

1.2.2.1.2 计算信用等级：属于功能服务

1.2.2.2.3 通知销售人员结果：属于应用服务，基础设施

⑤处理订单请求

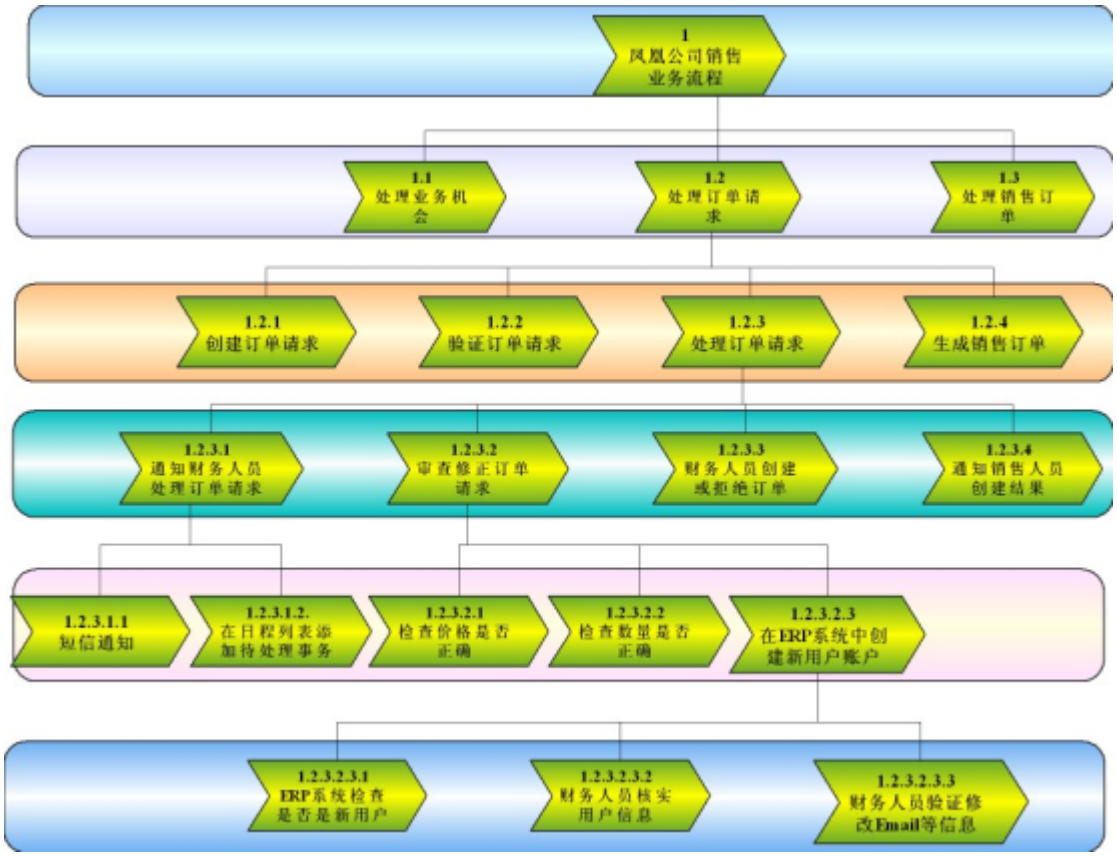


图 5 处理订单请求流程

1.2.3.1 通知财务人员处理订单请求：属于应用服务(短信通知服务和日程列表变更)

1.2.3.2 财务人员审查订单请求：属于人工服务

1.2.4 生成销售订单：属于应用服务

⑥出库流程（处理销售订单）

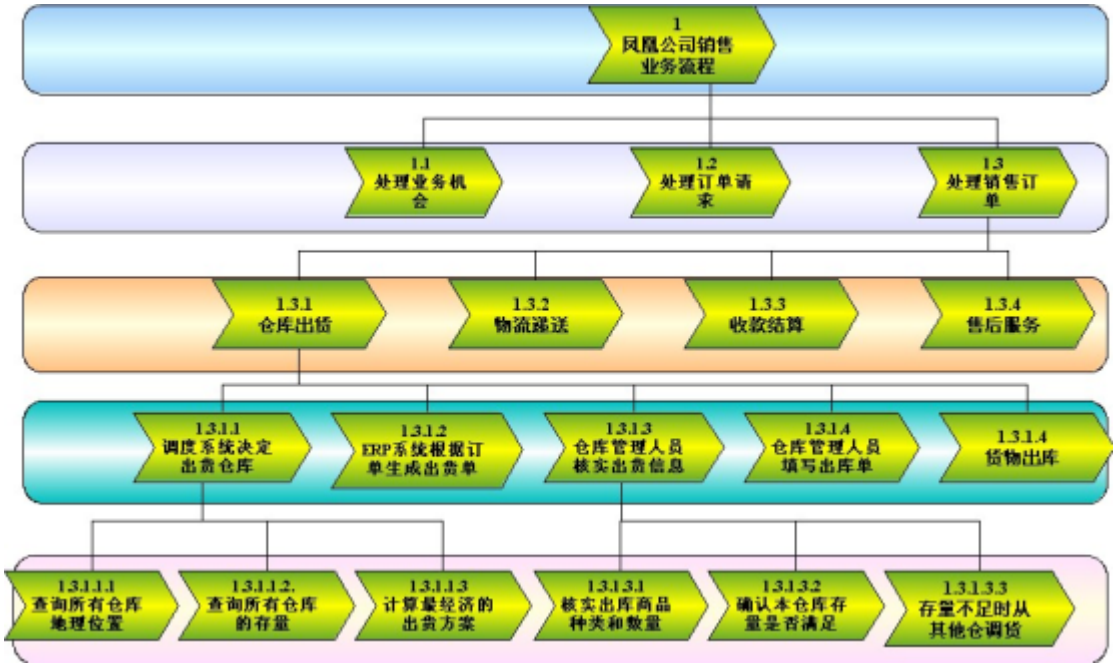


图 6 出库流程

1.3.1 仓库出库货：属于流程服务

1.3.1.1 调度系统决定出货仓库：属于应用服务

1.3.1.2 生成出货单（ERP 系统根据订单生成出货单）：属于应用服务

1.3.1.3.1/1.3.1.3（仓库管理人员核实）审查出库信息（核实出库商品种类和数量）：属于人工服务

1.3.1.3.2 确认本仓库存信息量是否满足：属于应用服务

1.3.1.4 仓库管理人员填写出库单：属于人工服务

⑦物流流程（处理销售订单）

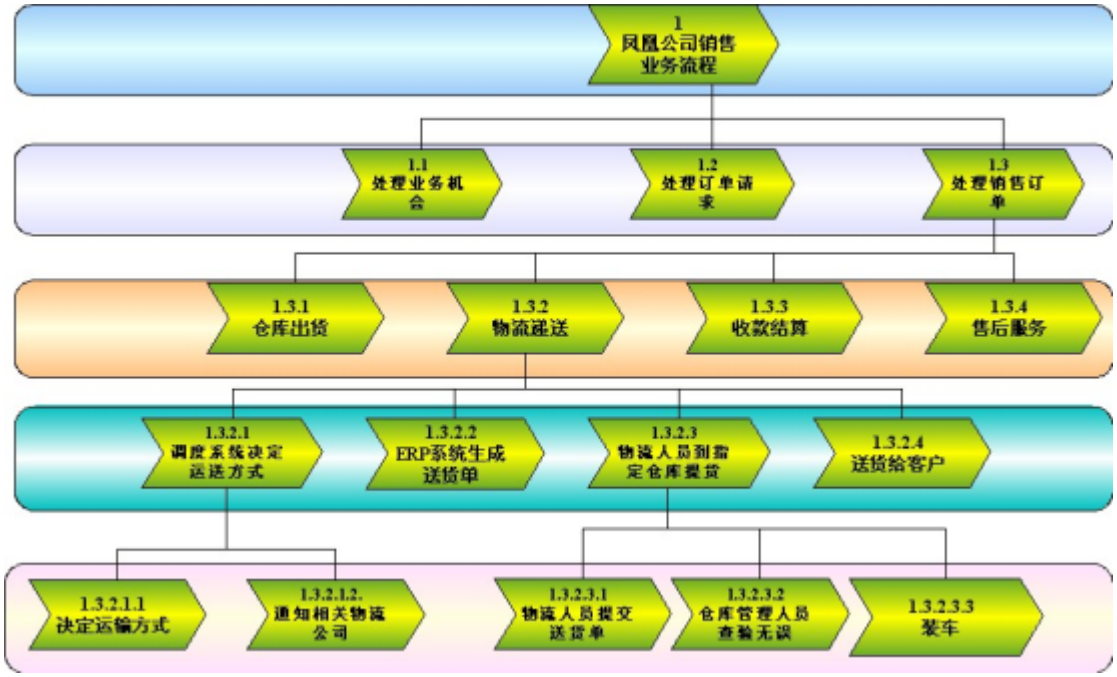


图 7 物流流程

1.3.2 物流递送：属于流程服务(外包)

1.3.3 结算：属于流程服务

1.3.2.1 调度系统决定运输方式：属于应用服务

1.3.2.2 ERP 系统生成送货单：属于应用服务

1.3.2.3.2 仓库管理人员查验无误：属于人工服务

1.3.2.3.3 取货、装车：属于人工服务

1.3.2.4 送货给客户：属于人工服务

⑧结算（处理订单请求）

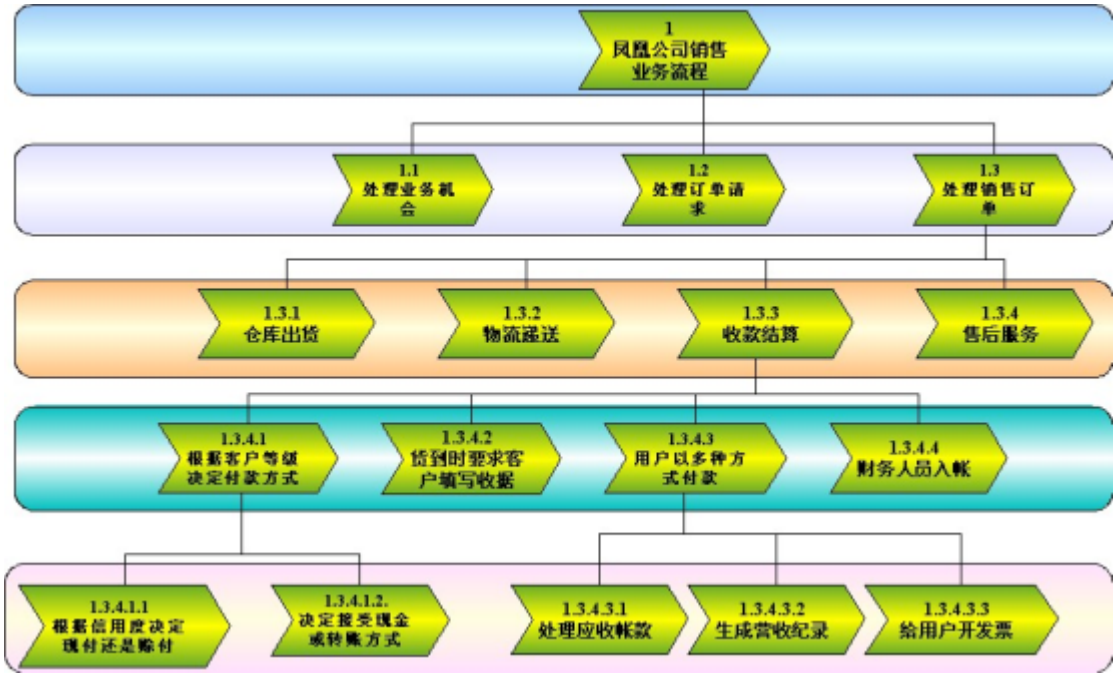


图 8 结算流程

1.3.4.1 根据客户等级决定选择付款方式：属于应用服务

1.3.4.3.2 生成营收纪录：属于应用服务

1.3.4.2 收据生成：属于应用服务

1.3.4.3.3 发票生成：属于应用服务

1.3.4.3.1 处理应收帐款：属于人工服务

1.3.4.3.3 给用户开发票：人工服务输入

1.3.4.3.1 通知付款：属于应用服务(通知系统)

1.3.4.4 财务人员入帐：属于应用服务

在上述所有提及的服务中，涉及规则的服务主要有以下内容:

①计算利润率和折扣率的算法规则;

②客户信用度评估;

③仓库调度选择;

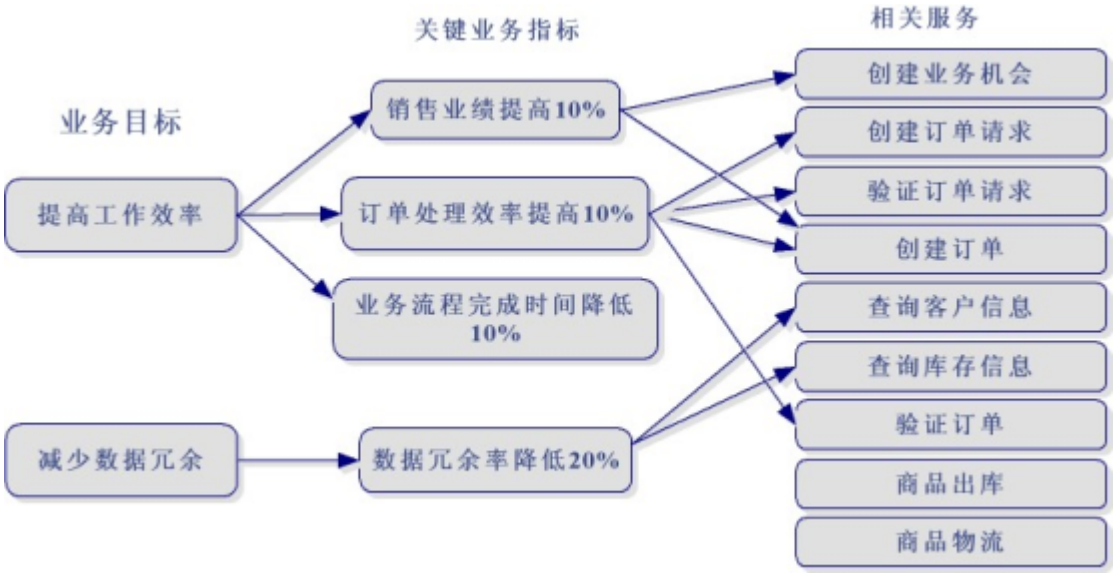
④物流调度选择;

⑤由客户等级决定付款方式;

1.2 服务发现的第二步：业务目标分解。

使用业务目标分解的分析方法帮助我们寻找可能的候选服务。我们的目标分解分为两种形式：一是关键业务指标的分解方式；一是在需求工程中常使用到的

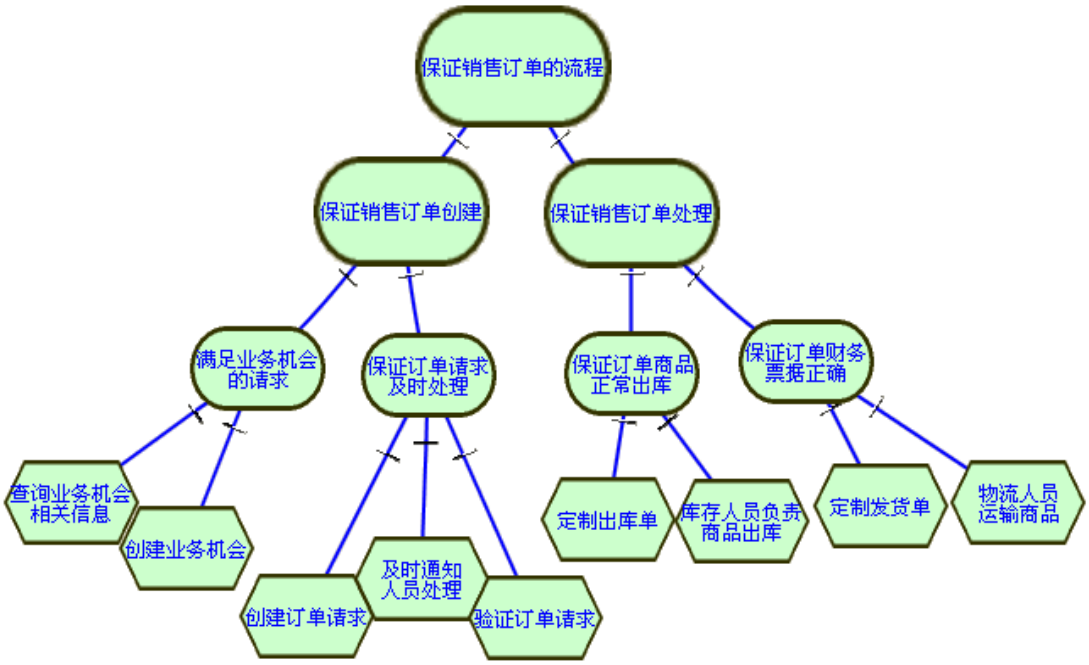
目标分析方法，对业务目标进行多层分解，把细粒度目标和一般的业务流程中的活动对应起来。第一种方法见下图：



目标分解图 1：关键业务指标分解图

通过这种分析方法，我们可能发现部分遗漏的候选服务。

第二种方法是面向目标的需求分析，如 GRL（Goal-oriented Requirements Language）。这是需求分析中十分重要的一种分析方法，通过目标分解的方法精化和细化我们的业务目标，能够清楚的把业务目标和所有的业务活动联系起来。在这里我们使用多伦多大学开发的建模工具 OME3 建立的 GRL 图如下：



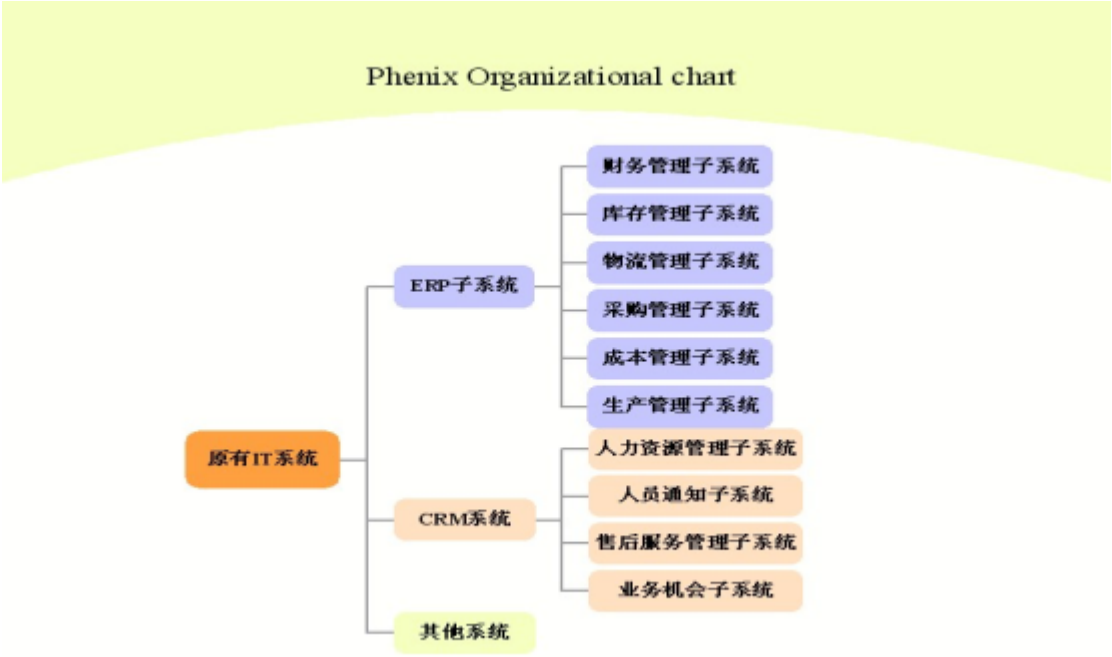
目标分解图 2：GRL 业务流程需求分析图

使用 GRL 来分析业务目标，优点在于能把非功能性目标如性能、安全等方面的

目标与功能性方面的目标一同考虑，在各种需求之间可以进行均衡的考虑。通过 GRL 的业务目标分解方法，我们对于候选服务有更全面的认识。

1.3 服务发现的第三步：遗留系统分析。

通过对于原有系统的分析，再次对我们系统中存在的服务遗漏进行补全，在凤凰公司中，主要是 ERP 系统和 CRM 系统，而我们可以按照下图的结构来分解原有 IT 系统:



在原有的系统中，基本包含了前面服务列表中的大部分的服务内容。

系统名称	对应服务
人力资源管理子系统	客户信息查询、修改、创建 查看客户信用记录等等
库存管理子系统	库存信息查询、修改、创建 商品查询、修改、创建 仓库选择调度 生成出库单
业务机会管理子系统	业务机会查询、修改、创建
成本管理子系统	计算利润率、计算折扣率、计算税率等 商品价格列表查询
人员通知子系统	人员通知服务
售后服务子系统	呼叫中心服务
物流管理子系统	选择运输方式、生成送货单

财务管理子系统	选择付款方式、生成帐目、通知付款

2. 服务目录确立

在上面服务发现确立了服务候选者的基础上，根据我们的考虑，将潜在的候选服务归类如下:

1 业务机会服务: 与业务机会相关的操作;

2 客户服务: 与客户信息相关的操作

3 库存服务: 与库存相关的操作;

4 目录服务: 查询信息相关的操作

5 销售订单请求服务: 与销售订单请求相关的操作;

6 销售订单服务: 与销售订单相关的操作

7 票据服务: 与票据相关的操作，主要票据有报价单、收据、出货单、发票、

8 呼叫中心服务: 与客户呼叫记录相关的操作，在这里主要是售后服务和客户定购的一种方式;

9 计算中心服务: 主要是计算操作，主要计算内容有利润率、折扣率、客户信用等级以及其他各类需要计算的操作;

10 帐目服务: 与帐目相关的操作，主要处理收款后的帐目记录、帐目变更;

11 通知服务: 通知各种人员，包括短信和日历记录通知;

12 调度服务: 负责与调度相关的操作，仓库选择调度和运输方式选择调度等调度相关操作;

上述服务是我们归纳的作为真正服务出现在 ESB (Enterprise Service Bus, 企业服务总线)中以及出现在流程服务中的应用服务，这些服务都是可重用的服务，他们是 SOA 思想的体现。在划分了服务之后，我们利用头脑风暴法把可能的服务都列举出来，并将他们按照对应 ESB 参考架构中的归位。如下所示:

创建业务机会: 流程服务

处理订单请求: 流程服务

处理销售订单: 流程服务

寻找商机: 人工服务

向客户报价: 人工服务

审核报价单: 人工服务

签订合同: 人工服务

财务人员审查订单请求: 人工服务

审查出库信息: 人工服务

填写收据: 人工服务输入

开发票：人工服务输入
填写出库单：属于人工服务
审核送货单：人工服务
取货：人工服务
运货：人工服务
通知付款：通知 service：应用服务(通知系统)
付款后更新帐目：billing service：应用服务
查看商品价格列表：目录信息 service 应用服务
计算利润率：计算利润率 service：功能服务，应用服务
计算折扣率：计算折扣率 service：功能服务，应用服务
客户呼叫：呼叫中心 service：应用服务，接收定购、接收投诉
创建业务机会记录：业务机会 service：应用服务
查看客户信息：客户信息 service：应用服务
查看库存信息：库存信息 service：应用服务
建立客户记录：客户信息 service：应用服务
更改业务机会状态：业务机会 service：应用服务
创建报价单：票据 service：应用服务
收钱帐目变更：billing service：应用服务
开收据：票据 service：应用服务
验证客户信用度：客户信息 service：应用服务
查看库存量：库存信息 service：应用服务
查看客户信用记录：客户信息 service：应用服务
计算信用等级：计算信用等级 service：功能服务
(规则判断是否同意：)
通知销售人员结果：通知 service：应用服务，基础设施
通知财务人员处理：通知 service：应用服务(短信通知服务和日程列表变更)
生成销售订单：销售订单 service：功能服务
仓库选择调度：调度 service：应用服务
生成出货单：票据 service：(自动生成出货单)应用服务
查看库存信息(是否满足)：库存信息 service：应用服务
调度决定运输方式：调度 service：应用服务
生成送货单：送货单 service：系统生成 应用服务
选择付款方式：billing service：应用服务
收据生成：票据管理 service：应用服务
发票生成：票据管理 service：应用服务

收款：billing service：应用服务

上述划分为我们后续的服务实现和随需应变的服务再组织打下基础，我们在实现时也会首先考虑上述服务是否能在我们的架构中完成。

SOA 中还有一些重要的包含规则的服务，在我们已经提到的服务中，下列服务 e 内容是包含规则，规则的制定将决定服务的结果：

1、计算利润率和折扣率的算法规则；

2、客户信用度评估评估；

3、仓库调度选择；

4、物流调度选择；

5、由客户等级决定付款方式；

上面一段大概的总结了有关服务列表的内容。

3. 服务规约

服务规约是对服务的属性、操作等基本性质的描述，我们在这里对所有已经列举的可重用服务进行了服务规约的描述。具体内容如下：

1、业务机会服务：

服务描述：与业务机会相关的操作，是整个订单流程的初始部分，销售人员一般先处理的都是业务机会，因此业务机会服务与销售人员的业务目标关联紧密；

2、客户服务：

服务描述：与客户信息相关的操作，是在一个企业中不论哪个子系统都使用频繁的一个功能模块，这个服务主要有 CRM 人员在维护，而销售人员和销售管理人员以及财务人员都会查看客户信息，同时，在订单自动处理的相关流程中，有些部分也需要核对客户信息，因此客户服务可以说是一个基本功能的服务，与一般业务目标和业务流程相关。

3、库存服务：

服务描述：与库存相关的操作，库存服务覆盖了库存中商品信息和出入库管理，而商品信息是企业的一个基础成分，在销售中也是属于基本信息；而出入库管理是销售流程中的必经部分，对于库存服务来说，一是维护了企业的基本商品信息，直接与销售情况相关，二是管理了出入库行为，直接影响物流和业务机会能够建立。

4、目录服务：

服务描述：查询信息相关的操作，这里主要指的是销售人员操作时，需要查看商品信息列表和商品价格列表信息，商品信息来源于库存服务，商品价格信息

来源于财务人员的议定，目录服务强调的是销售人员查看信息来向客户提供信息和报价等等，属于业务流程中需要的功能服务，直接关系到业务是否能够成功。

5、销售订单请求服务:

服务描述: 与销售订单请求相关的操作，销售订单请求来源于业务机会信息，目标是请求创建销售订单，从而开始一个订单处理过程。这个服务是在这些服务中比较特殊的，因为它可以不单独存在，因为我们可以不把销售订单请求当作实体，而只是要求创建订单的一种行为。但是这里我们为了清楚的表示出订单流程，单列了订单请求这个服务，这个服务是在业务机会创建之后，销售人员要求创建销售订单的载体，销售订单请求中的信息是销售订单中信息的主要部分。在订单处理这个流程中，订单请求服务是有必要存在的。

6、销售订单服务:

服务描述: 与销售订单相关的操作，销售订单创建之后即使表示这次销售行为成功，会转到库存和物流服务来处理剩余部分流程。销售订单的信息反应了这次销售行为的所有细节，可以按照这个销售订单来生成出货单，交给库存服务来继续后面工作。销售订单服务是订单处理流程的核心，必然是不能少的。

7、票据服务:

服务描述: 服务与票据相关的操作，主要票据有报价单、收据、出货单、发票等。票据服务可以说是为了使流程自动化的一种服务，不论是哪种票据的生成，都是通过一种实体信息输入，比如出货单的输入信息是销售订单。通过票据服务的存在，我们流程中存在的多种票据都能自动化的产出，不会有过多的人工要求。

8、呼叫中心服务:

服务描述: 呼叫中心是很重要的企业组织部分，在这里强调的是与客户呼叫记录相关的操作，主要是售后服务和客户定购商品的一种方式; 用户通过呼叫中心要求购买商品，呼叫中心会添加服务记录，然后转发给销售人员，销售人员再开始业务机会服务。

9、计算中心服务:

服务描述: 主要是计算操作，主要计算内容有利润率、折扣率、客户信用等级以及其他各类需要计算的操作; 这些计算结果是将分布到不同流程中的，是流程中不可或缺的数据，我们在这里把所有与计算相关的功能组件集中在这一个计算中心服务里。

10、帐目服务:

服务描述: 与帐目相关的操作，主要处理收款后的帐目记录、帐目变更; 财务人员在收款等操作后，企业原有的帐目、营销记录需要更新，通过这个服务自动完成帐目更新的过程，帐目清晰对于企业的财务是十分重要的，所以这个服务也是很重要的。

11、通知服务:

服务描述: 在发生某些事件或是请求后，需要通知某人对应人员来处理请求，这就需要通知服务。在销售订单请求发生时，需要通知财务人员来处理，这时就需要通知服务。

12、调度服务:

服务描述: 负责与调度相关的操作，仓库选择调度和运输方式选择调度等调度相关操作；在仓库需要出货时，我们的调度服务就需要选择出库的仓库。这个只是调度服务的一个方面，还可能存在许多的调度相关的操作，在实现的时候可以补充进来。

下面列表给出与服务相关的属性和服务操作信息。

表 1. 应用服务对应规约表

服务名称	**关键服务属性**	**服务操作**
业务机会服务	业务机会记录信息 业务机会状态 （业务机会名称 客户 客户联系人 负责的销售人员 对应商品信息 业务机会状态 业务机会描述）	1 创建业务机会记录 2 修改业务机会记录(修改业务机会状态) 3 删除业务机会记录 4 查看业务机会记录(通过各种属性查看对应业务机会) 5 查看业务机会状态
客户服务	客户信息 （ 客户名称 客户联系人 性别 单位 职务 电话 客户等级(高或低) 客户备注信息 客户信用等级 客户信用记录 ）	1 创建客户信息 2 修改客户信息 3 删除客户信息 4 查看客户信息(通过各种属性查看对应客户信息) 查看客户信用记录
销售订单请求服	订单请求信息	1 根据业务机会创建订单请求

务	（ 业务机会信息 订单请求创建人 订单请求创建时间 订单请求描述 ）	2 查看订单请求信息 3 删除订单请求信息 4 修改订单请求信息
销售订单服务	销售订单信息 （ 客户负责人 销售人员 客户联系人 合同状态(创建、审核、出库、完成) 销售商品列表 订单签订日期 商品金额 其他费用 备注 订单创建日期 提交人 用户行为(订货&退货) ）	1 查看销售订单信息 2 审核销售订单信息 3 增加销售订单信息 4 修改销售订单信息 5 删除销售订单信息 6 修改对应销售商品信息
通知服务	待通知人员信息 通知信息 日历信息 反馈信息(成功&失败)	1 通知事务 2 修改人员日程信息 3 查看反馈信息
目录信息服务	商品信息列表 商品价格列表	1 查看商品信息列表 2 修改商品信息 3 增加商品信息 4 删除商品信息 5 查看商品价格列表 6 修改商品价格信息 7 增加商品价格信息 8 删除商品价格信息
库存服务	库存商品信息	1 查看库存商品信息(根据各种

	(名称、类型、规格、单位、价格、库存数量) 出入库记录 (对应商品，出入库商品价格、出入库数量、库单标记)	属性查看库存商品) 2 修改库存商品记录 3 删除库存商品记录 4 增加库存商品记录 5 增加、删除、修改、查看出入库记录
计算功能 service	折扣率 利润率 客户信用等级 其他主要需要计算的相关单位	输入对应参数 计算折扣率、利润率等等
票据服务	票据类型 票据信息 （ 不同类型票据不同信息 ）	根据输入信息对应生成 1 报价单 2 收据 3 出货单 4 发票
呼叫中心服务	呼叫中心服务记录 （ 客户 要求服务内容 ）	1 记录客户服务记录 2 转发到对应部门处理
帐目服务	营销帐目记录	1 增、 2 删 3 改 4 察帐目记录 5 根据收款录入情况更新帐目
调度服务	仓库调度信息	选择对应出库的仓库

上面提到过的服务属性、服务操作等等规约描述是我们在实现服务时需要主要考虑的因素。对于任意一个服务来说，规约是能唯一标识这个服务的信息。

4. 服务实现

在完成了服务的规约之后，我们就要开始实现服务了，服务实现有三种方式：服务包装、新服务开发、服务中介。下面分别介绍。

4.1 服务包装

遗留系统中已有的应用和信息是实现业务逻辑和业务数据的重要资产。通过集成已有的应用和信息将可以在企业系统上实现更多增值服务，集成已有应用和信息是企业集成中重要的一环。而我们对于原有应用的处理的主要方式是将原有应用包装成服务，在通过包装的方式实现这些服务以后，我们会把服务映射到 ESB 服务总线上去。

ESB 通过各种适配器技术将已有系统中的业务逻辑和业务数据包装成企业服务总线支持的协议和数据格式。通过企业服务总线，这些被包装起来的业务逻辑和数据就可以方便的参与上层的业务流程，从而已有应用系统的能力可以得以继续发挥。实际上，在我们的服务实现中，应该尽可能多的发掘和使用原有系统中的应用内容，大量的重用原有的系统应用能减少维护费用、开发费用以及开发时间，也是 SOA 的优势所在。

在凤凰公司这个整合过程中，我们现在提到的所有的应用服务中，都可以通过对原有系统的包装来完成服务实现。

服务名称	**利用原有系统应用实现**
业务机会服务	封装原有 CRM 系统中的 业务机会处理的系统应用
客户服务	封装原有 CRM 系统中的 客户信息相关的系统应用，
库存服务	封装原有 ERP 系统中的 库存管理相关的系统应用
目录服务	封装原有 ERP 系统的 1、商品价格系统查询应用和 2、商品信息查询应用
销售订单请求服务	封装原有 ERP 系统中 销售订单请求相关的应用
销售订单服务	原有 ERP 系统中的销售订单相关应用
票据服务	封装原有 ERP 不同系统中 1、报价单生成应用 2、收据生成应用 3、出货单生成应用

	4、发票生成应用
计算中心服务	封装原有 ERP 成本管理子系统中 1、计算利润率应用 2、计算客户信用等级应用 3、计算客户折扣率应用
帐目服务	封装原有 ERP 财务子系统中的 帐目管理应用
通知服务	封装原有 CRM 系统中的 通知应用
调度服务	封装原有 ERP 系统中的 仓库调度选择应用

服务包装实现列表

4.2 新服务编写

在服务实现中，新服务的编写是不能避免的，但应该说一般希望能尽量多的重用原有系统，而避免开发新的服务。

新服务的编写主要集中在流程服务中，对于所有的以细粒度服务组合成的粗粒度服务，都是需要在流程服务中体现的，流程服务以一定的顺序将其需要的细粒度服务组合起来，对服务请求者提供了一个更大粒度的服务。

在我们这里，主要的流程服务如下：

4.2.1 创建业务机会

4.2.2 处理销售订单请求

4.2.3 处理销售订单

4.2.4 报价

4.2.5 验证订单请求

4.2.6 仓库出库

4.2.7 物流递送

这些流程服务并不完整，我们希望达到的目的是，在新的流程需要的时候，我们能够很快的组织一个新的流程服务来满足这个需求，以达到随需应变的 SOA 应用的目标。

这些流程服务的编写是有充分的工具支持的，典型的流程开发工具就是 BEPL。

我们分析开发“创建业务机会”这个流程服务的过程。

创建业务机会包括的业务流程如下：

①销售人员寻找商业机会；

②商机出现时，销售人员查询客户帐户看是否有这个帐户存在，没有则创建新帐

户；

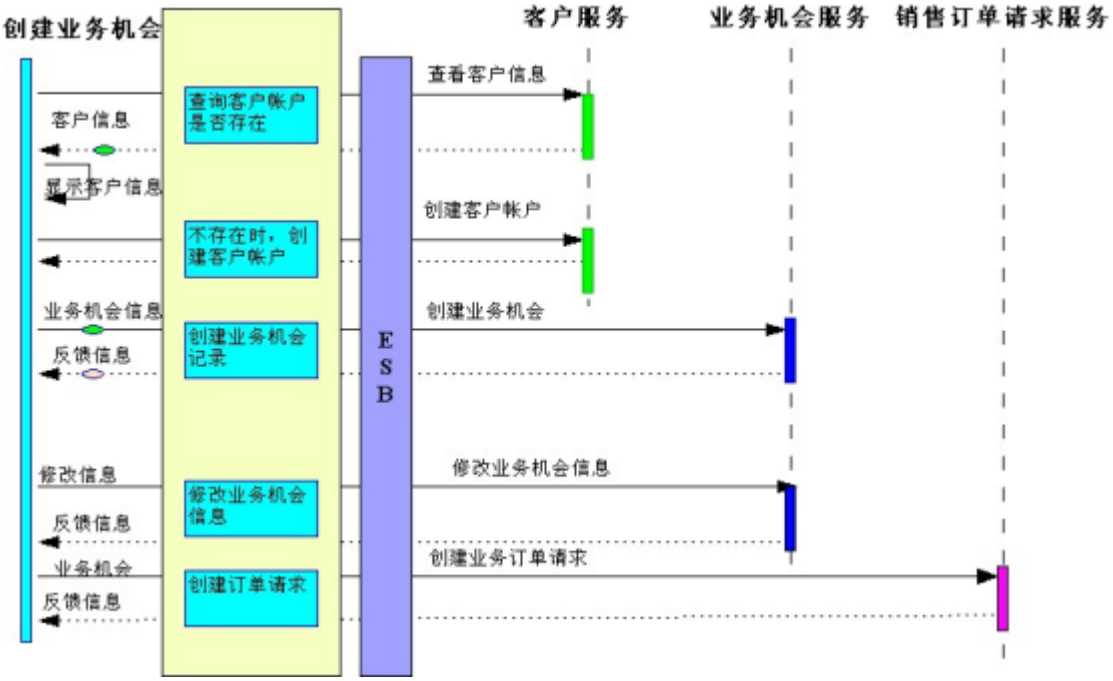
③创建业务机会记录；

④和客户谈判过程中，修改业务机会记录信息；

⑤修改业务机会状态为“赢”；

⑥创建销售订单请求；

这个流程中使用的细粒度服务主要有客户服务、业务机会服务、销售订单请求服务；我们以下图来说明这个流程中细粒度服务的组成关系：



这个图反映了服务与客户行为之间的交付。

在开发这个流程服务时，我们将把服务按照上述过程组织起来，对外就形成了一个创建业务机会的流程服务。

开发新服务这种服务实现方式，针对的主要是流程服务，在新的流程出现时，我们的新服务开发为先分析处流程服务中涉及的业务流程，寻找这些流程中的服务内容，在把这些服务按照流程顺序组织起来，就能完成一个新服务的开发。

实现流程开发的工具是 BEPL。

4.3 服务中介

服务实现的另外一种方式则是使用服务中介，这种实现方式中，多是在使用外部提供的服务时使用。ESB 借助于服务注册管理以及问题域相关的知识（如业务方面的一些规则等）自动进行的，不需要服务请求者和提供者介入，从而实现了解耦服务请求者和提供者的技术基础，使得服务请求者不需要关心服务提供者的位置和具体实现技术，双方在保持接口不变的情况下，各自可以独立地演变。

在我们这里的服务中，主要是物流服务和呼叫中心服务两个服务是属于这种

实现情况，另外，银行在企业中扮演的角色也常常是这种外部服务提供商，下面分析利用服务中介方式实现的服务:

①呼叫中心服务：呼叫中心在企业中实现的成本很高，并不适合在企业内部开发和维护，于是多采用外包的形式，把呼叫中心外包给专业的公司来做，呼叫中心负责这个呼叫中心的开发和维护，最后通过适配器的方式来整合到 ESB 总线中，形成一个松耦合的状态。

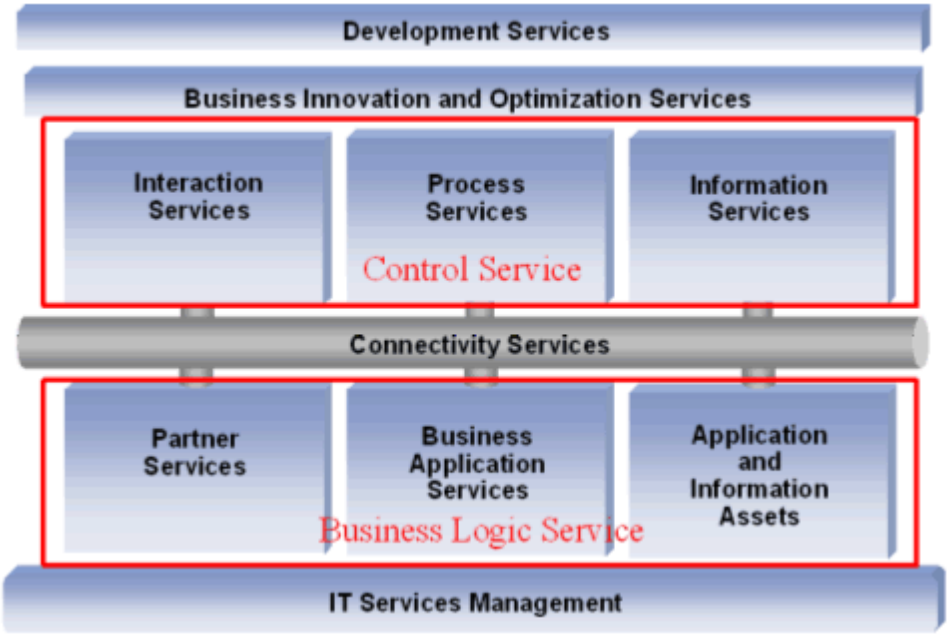
②物流服务通常作为公司来说也是外包给专业的物流人员来完成，物流服务中需要处理的部分也会通过适配器和总线整合起来。

外包是服务中介的主要的存在方式，因为存在外包的服务和实现上的不同接口，我们需要服务中介来处理外包了的服务和总线之间的差异。

上面阐述了服务实现的三种方法，服务包装、新服务开发以及服务中介，每种服务实现方式有其特定适合的场合，我们需要根据我们需要实现的服务来有针对性的选择实现方式来实现服务。

4.4 服务实现的 ESB 映射

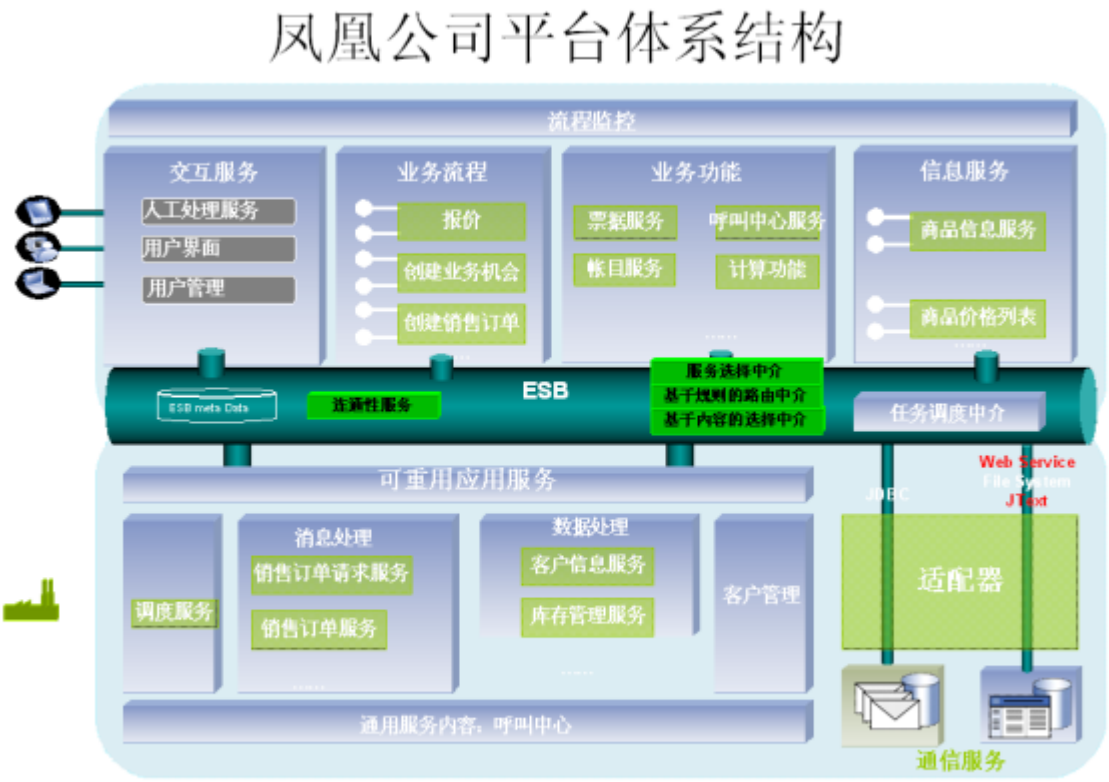
ESB 本身是有一个参考架构的，这个参考架构如下图所示:



ESB 参考架构

图中的对应 service 分块是前面我们提到过的服务分类，通过把我们定义好的服务归纳分类，放到不同的服务类型中，就把我们的服务映射到 ESB 结构中，在实际实现中，我们会使用 IBM 对应的 webspere 工具帮助我们完成 ESB 总线和各个服务的实现。

我们能把所有的服务都映射倒 ESB 的参考结构中去，这个映射结果如下图所示:



这个结构并不是最终结构，而是工作的一个参考，我们会实现了模块都按照 ESB 的实现方式和总线连接起来，形成完整的体系。

5. 总结

到这里，我们完成了服务的相关基本工作，大致上来说是完整的，但是对于许多细节，我们是做的不够的，所以在实现系统的时候必然会对服务有更加清楚的描述。

作为 SOA 来说，最基础的一个内容就是服务，对于服务的定义、服务的规约描述、服务的实现都是会影响全局的工作，这个文档依次对服务的发现、规约、实现进行了阐述，希望能对服务的概念有清楚的展示，对服务思想有清楚的表达。

组件设计

引言

面向服务的体系结构（service-oriented architecture，SOA）本身是一个组件模型，它将应用程序的不同功能单元（称为服务）通过这些服务之间定义良好的接口和契约联系起来。接口是采用中立的方式进行定义的，它应该独立于实现服务的硬件平台、操作系统和编程语言。这使得构建在各种这样的系统中的服务可以以一种统一和通用的方式进行交互。

组件是组织服务的结构，从组件的状况可以看到服务的分布和部署等情况。组件是实现工作的基础，定义良好的组件将为服务定义和实现打好基础，在组件设计的过程中，我们按照下列的步骤来完成:

① 总体组件划分和组件功能描述

② 组件接口定义

③ 具体实现分析

④ 系统实现结构和数据流程

组件的分析过程主要在于组件的接口定义，这些接口将决定组件的使用方式，从而决定组件的可复用程度。组件可复用的程度是对组件本身性能一种最直接的体现。

下面将对我们的组件分析过程进行描述。

1. 总体组件划分

对于一个一般的 IT 系统来说，组件划分定义是最关键的一步，因为组件关系到系统实现中的分工与接口统一，合理的组件定义能使整合系统所花费的时间减少，而冗余、烦杂的组件定义会使系统难于整合，组件之间关系不清晰，导致开发的困难。

相比一般的 IT 系统实现，在 SOA 中，我们的组件划分应该考虑的问题相对更多，我们需要把组件与服务联系起来，同时又应该有合理的区分，在我们的组件定义中，组件是相关服务的整合，多个功能相关的服务组成一个组件。

对于系统的组件划分应该遵循一些基本原则:

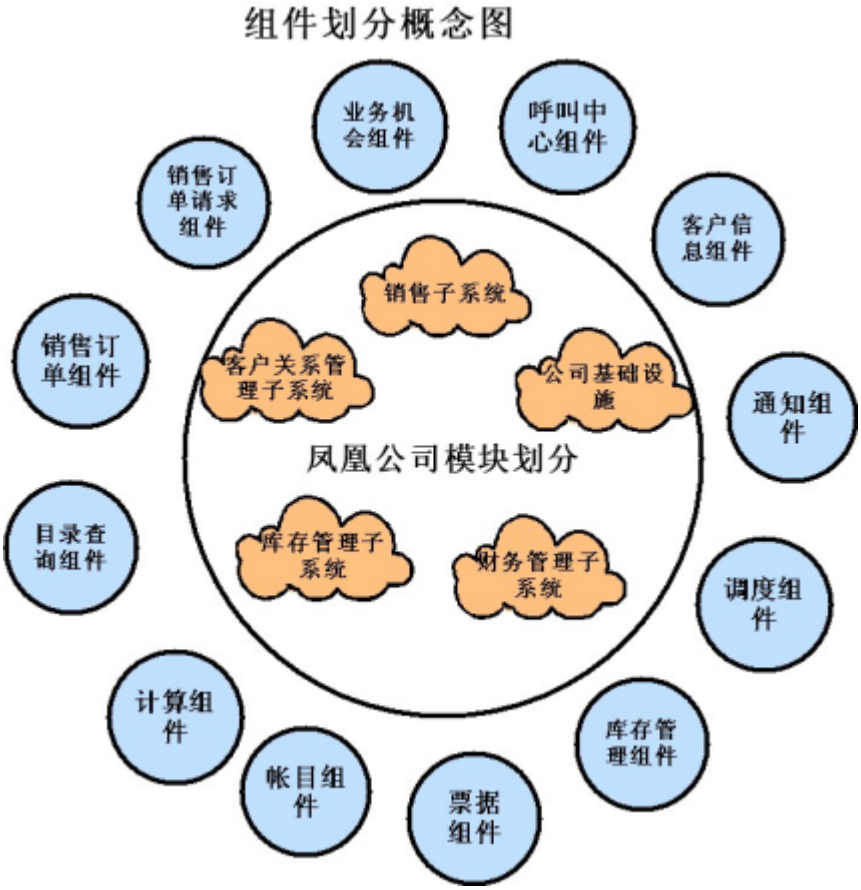
①可复用。应该使定义的组件能够满足尽可能多的复用。

②功能完整。每个组件应该能独立完成功能，这种清晰的功能定义使得开发人员能够清楚理解组件。

③接口清晰。每个组件应该有清晰的输入和输出，使得流程开发人员和整合开发人员能够准确的使用

④在 SOA 中有比较特殊的要求，即服务的体现。在组件中应该能看到细粒度服务的存在。

整个的功能组件划分如下图所示:



组件划分图

组件是组织服务的形式，从组件的划分形式可以看到我们大概的实现思想，上面提到的组件都是可重用的组件。

下面我们将逐一分析每一个组件的功能，组件的功能是由其对应封装的服务决定的，所以对应组件一般是封装一些关系比较紧密的服务。我们在服务分析与设计文档中定义了服务，可以对比两个文档，看出服务在组件定义中的体现。下面对各个组件的功能进行一些描述:

1.1 业务机会组件:

1.1.1 基本描述: 业务机会相关，在销售处理过程中业务机会是最先使用的组件

1.1.2 功能描述: 创建、修改、删除、查看业务机会

1.1.3 备注: 业务机会中有业务机会状态的属性;

1.2 客户组件:

1.2.1 基本描述:

管理客户信息，是 CRM 系统的基础组件，在多个流程和服务中都会使用到;

1.2.2 功能描述:

创建、修改、删除、查看客户信息

查看客户信用记录

查看客户联系人信息

1.2.3 备注：客户信用记录在计算组件中会使用到。

1.3 库存组件：管理库存相关的信息和出入库操作

1.3.1 基本描述:

库存信息管理，包括库存商品的信息以及出入库管理，在订单处理流程中会使用到;

1.3.2 功能描述:

创建、修改、删除、查看库存商品信息;

创建、修改、删除、查看出入库记录;

1.3.3 备注：库存商品信息会在目录组件中被销售人员查询使用。

1.4 目录信息组件:

1.4.1 基本描述:

主要是查询功能，销售人员在创建销售机会时，需要查询商品的信息列表；在向客户报价时，需要查询商品的价格列表;

1.4.2 功能描述:

查看商品信息列表;

查看商品价格列表;

1.5 销售订单请求组件

1.5.1 基本描述:

销售订单请求是一个比较特殊的组件，原本可以和销售订单合成为一个组件，这里将它单列出来，描述销售订单创建前有一个销售人员的订单请求过程;

由销售人员创建，订单请求的数据来源于业务机会相关的信息;

1.5.2 功能描述:

创建、修改、删除、查看销售订单请求的信息;

1.5.3 备注：由销售人员创建，给销售管理人员检查后再生成销售订单

1.6 销售订单组件:

1.6.1 基本描述:

销售订单是后续是生成出库单，销售订单由销售管理人员创建，描述了业务机会的具体信息，前身是销售订单请求;

1.6.2 功能描述:

创建、修改、删除、查看销售订单的信息;

1.6.3 备注：销售订单和其他单据的关系紧密。

1.7 票据生成组件:

1.7.1 基本描述:

在整个系统中的票据有许多种，这里主要有报价单、收据、出货单、发票；这些票据是可以自动化的由输入产生对应票据输出的，所有会有票据这个组件。

1.7.2 功能描述:

根据输入的信息生成标准格式的报价单、收据、出货单、发票;

1.7.3 备注：这些票据的种类在开发中，我们可以根据需要变更，不一定就是这几种。

1.8 呼叫中心组件:

1.8.1 基本描述:

与客户呼叫记录相关的操作，在这里主要是售后服务和客户定购的一种方式;

1.8.2 功能描述:

创建、修改、删除、查看呼叫中心记录的信息;

1.8.3 备注：呼叫中心是完整企业必备的一个系统，它接收外界的呼叫，主要是售后服务部使用;

1.9 计算中心组件:

1.9.1 基本描述:

计算中心处理的是与计算相关的大多数功能组件，这些组件内部的规则针对计算内容设计，主要计算内容有利润率、折扣率、客户信用等级以及其他各类需要计算的操作;

1.9.2 功能描述:

计算选择的对应内容，输入数据，输出数据

1.9.3 备注：在实际实现时，可能考虑更多的计算内容;

1.10 帐目组件:

1.10.1 基本描述:

与帐目相关的操作，主要处理收款后的帐目记录、帐目变更;

1.10.2 功能描述:

创建、修改、删除、查看帐目记录的信息;

1.10.3 备注：一般在收款以后，会有帐目变更

1.11 通知组件:

1.11.1 基本描述:

通信功能，可以说是系统的基础服务，向对应人员通知信息

1.11.2 功能描述:

通知对应人员对应消息

1.11.3 备注：通信对象可以有很多类型

1.12 调度组件：负责与调度相关的操作，仓库选择调度和运输方式选择调度等调度相关操作；

1.12.1 基本描述：

调度选择出库仓库，有一个选择规则在其中起关键作用

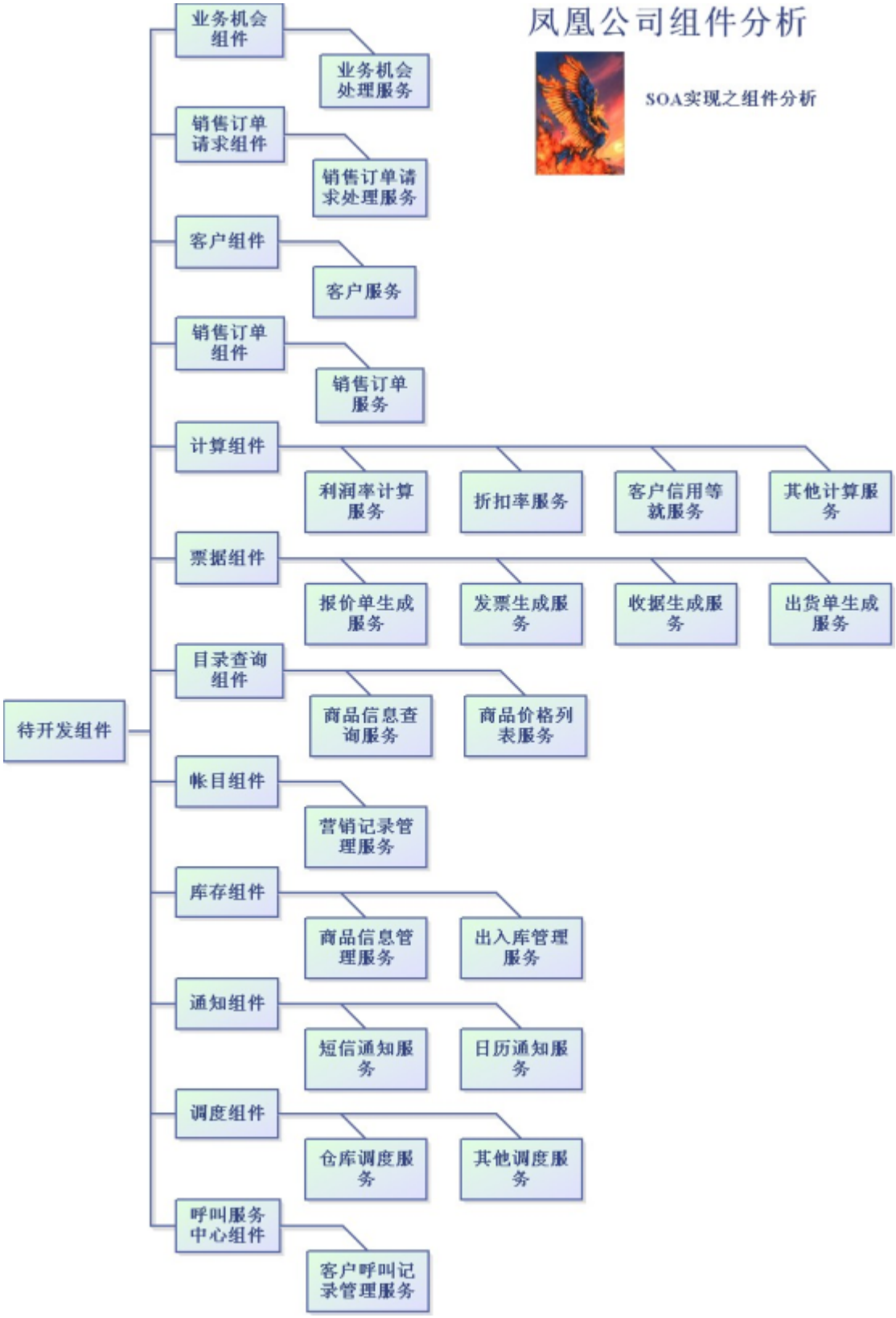
1.12.2 功能描述：

为某一输入出库单选择出库仓库

1.12.3 备注：选择算法很关键

2. 组件接口定义

在确定了各个功能组件的基础上，我们把各个组件按照它所对应的功能组织服务，每个服务将对应一个完整的功能，下图从整体上反应这个功能组件的分布。从这里直接体现了服务与组件的关联。



下面给出的是组件的接口定义和对应的功能描述信息。

组件	**组件功能**	**输入信息**	**输出信息**
业务机会处理组件	创建、修改、删除、	需要创建、修改或是	被修改的业务机会信

	查看业务机会	查看的业务机会信息内容	息，反馈信息
销售订单请求处理组件	创建、修改、删除、查看销售订单请求的信息;	需要创建、修改或是查看订单请求的业务机会信息	反馈信息或是销售订单请求信息
销售订单处理组件	创建、修改、删除、查看销售订单的信息;	需要创建、修改或是查看的销售订单请求的信息	反馈信息或是销售订单信息
客户信息管理组件	创建、修改、删除、查看客户信息; 查看客户信用记录; 查看客户联系人信息	需要创建、修改或是查看的客户信息	反馈信息或是客户对应信息
计算组件	计算相关工作，包括利润率、折扣率等等	计算利润率需要的基本输入，包括税率等等	计算出的结果
票据组件	根据输入的信息生成标准格式的票据	需要生成票据的信息	生成的票据
目录信息组件	查看信息列表	输入查询要求	符合要求的信息列表
帐目管理组件	根据销售订单更新帐目记录的信息;	销售订单信息	帐目更新后的状态
库存管理组件	创建、修改、删除、查看库存商品信息、创建、修改、删除、查看出入库记录	需要修改的商品信息 库存管理人员填写的出库单信息	反馈信息或是被查看的商品信息 被增加的出入库记录信息
通知组件	通知对应人员对应消息	通知信息、对应人员标识信息	发出的反馈信息、是否成功
调度组件	选择出货仓库	出库单信息	选择的出库仓库
呼叫中心组件	创建、修改、删除、查看呼叫中心记录的信息;	呼叫记录信息，可能是售后服务要求，可能是订单要求等等	添加的呼叫中心记录

组件功能分析及接口定义表

在上述表格中，我们定义了各个组件的接口，在发生组件调用时，我们需要使用的就是这些接口。在一个完整的流程中，会有很多自动化的组件和服务调用，它们调用的基本内容就是其他组件的接口，定义良好的接口使得开发相对简单。

3. 具体实现分析

对于具体的实现，我们始终觉得更多的应该考虑原有系统状况，在原有 ERP 和 CRM 的基础上多考虑原有系统的复用，能够减少开发量和开发时间。

在我们的组件实现中，我们主要考虑两种方式:

①原有系统功能封装成组件。将原有 ERP 和 CRM 系统中提供的功能接口按照我们需要的组件功能封装起来，同时对于使用组件的人员是透明的。

②针对功能开发新的组件。有的功能原有 ERP 和 CRM 没有涉及，因此我们可能需要开发部分新组件。

多数情形下我们只需要考虑封装原有系统就能完成组件，而且出于开发的考虑，我们也会尽量避免不必要的新组件开发。

下面解释两种组件实现。

3.1 组件封装

遗留系统中已有的应用和信息是实现业务逻辑和业务数据的重要资产。通过集成已有的应用和信息将可以在企业系统上实现更多增值服务，集成已有应用和信息是企业集成中重要的一环。在原有 CRM 和 ERP 中已经包含了我们需要的绝大多数的功能，我们需要做的只是使用原有 ERP 和 CRM 的接口，按照我们需要的方式组织接口顺序，最后对外提供组件对应的接口。

我们现在提到的组件，可以通过对原有系统的封装来完成组件实现。

组件名称	**利用原有系统应用实现**
业务机会处理组件	封装原有 CRM 系统中的业务机会处理的系统应用
客户信息管理组件	封装原有 CRM 系统中的客户信息相关的系统应用，
库存管理组件	封装原有 ERP 系统中的库存管理相关的系统应用
目录信息组件	封装原有 ERP 系统的 1、商品价格系统查询应用和 2、商品信息查询应用
销售订单请求处理组件	封装原有 ERP 系统中销售订单请求相关的应用
销售订单处理组件	原有 ERP 系统中的销售订单相关应用
票据组件	封装原有 ERP 不同系统中 1、报价单生成应用 2、收据生成应用 3、出货单生成应用 4、发票生成应用
计算组件	封装原有 ERP 成本管理子系统中

	1、计算利润率应用 2、计算客户信用等级应用 3、计算客户折扣率应用
帐目管理组件	封装原有 ERP 财务子系统中的 帐目管理应用
通知组件	封装原有 CRM 系统中的 通知应用
调度组件	封装原有 ERP 库存子系统中的 仓库调度选择应用

组件封装实现列表

上述列表清楚概述了通过封装原有 ERP 和 CRM 系统中的接口来实现组件功能的方法。在具体实现组件时，我们会具体分析对应 ERP 和 CRM 系统已经提供的功能和其对应的接口，把那些能够提供完整功能的部分作为组件直接提炼出来，比如，对于客户管理组件，CRM 系统中应该有一套完整的应用程序，我们把这些应用按照我们描述的接口提供给使用者，完成对组件的封装。而另外一些原有系统以分布式形式提供功能的应用，我们把他们整合成我们希望看到的接口形式，比如，对于计算组件，可能包含了来自原有系统的不同地方的应用程序，我们就会按照计算组件的接口形式，把所有应用程序都封装成我们期望的接口，完成组件的封装。

不管具体怎么样做，组件封装主要的内容来自于原有系统，而对于原有系统的理解就将决定我们组件编写的好坏。所以，在实际实现时，我们需要对原有系统进行清楚的分析。

3.2 新组件编写

在实现中，我们其实是在尽量复用原有系统的应用程序，而避免自己开发新的组件，但是开发新组件也是必要的。在我们目前的组件划分中，没有明显的需要完整开发的新组件，全部都是原有系统中提供了大部分功能的组件，但是在两个系统集成中可能有我们没有考虑到的功能需求和非功能性需求，这些需求就需要我们开发新的组件，通过开发新的组件来满足这些需求；另外，如果系统需要新增功能，也就可能需要开发新的组件。

总体来说，我们希望能尽可能的少使用新开发的方式来实现组件，在目前的组件划分中，并不需要通过重新编写来实现组件，但是随着系统的变更，我们可能需要开发新的服务，开发新的组件。

3.3 组件与服务

我们应该注意到，组件实现中的服务实现问题。针对 SOA 来说，当出现一

个业务需求的时候，我们可能要面对几种可能:

①不需要开发新的应用服务，也不需要开发新的流程服务；只是把已有的可重用服务按照已有流程形式组织起来，即可满足业务需求。

②不需要开发新的应用服务，但是需要开发新的流程服务；业务需求是新的流程形式，需要做的是的原有应用服务组织成新的流程，这个新的流程服务能够满足这个业务需求。

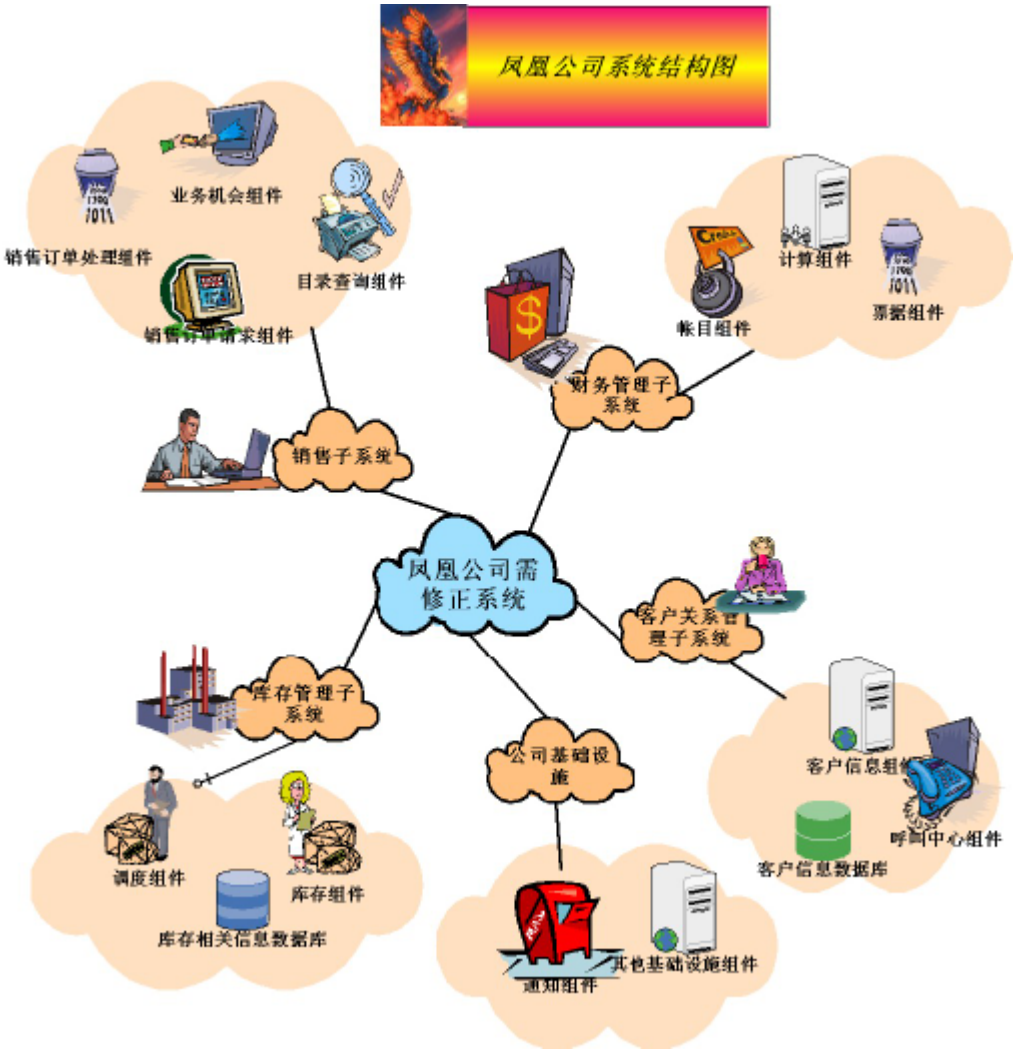
③需要开发新的应用服务，同时也需要开发新的流程服务；业务需求的流程和细粒度服务都需要新开发，在开发新的细粒度应用服务之后再组织成一个新的业务流程。

当出现新的应用服务开发时，我们的组件必然会对应出现变化，可能需要修改原有组件，可能需要新增组件。

服务与组件是密切相关的，在 SOA 随需应变的思想下，必然会有服务、组件的灵活变动。

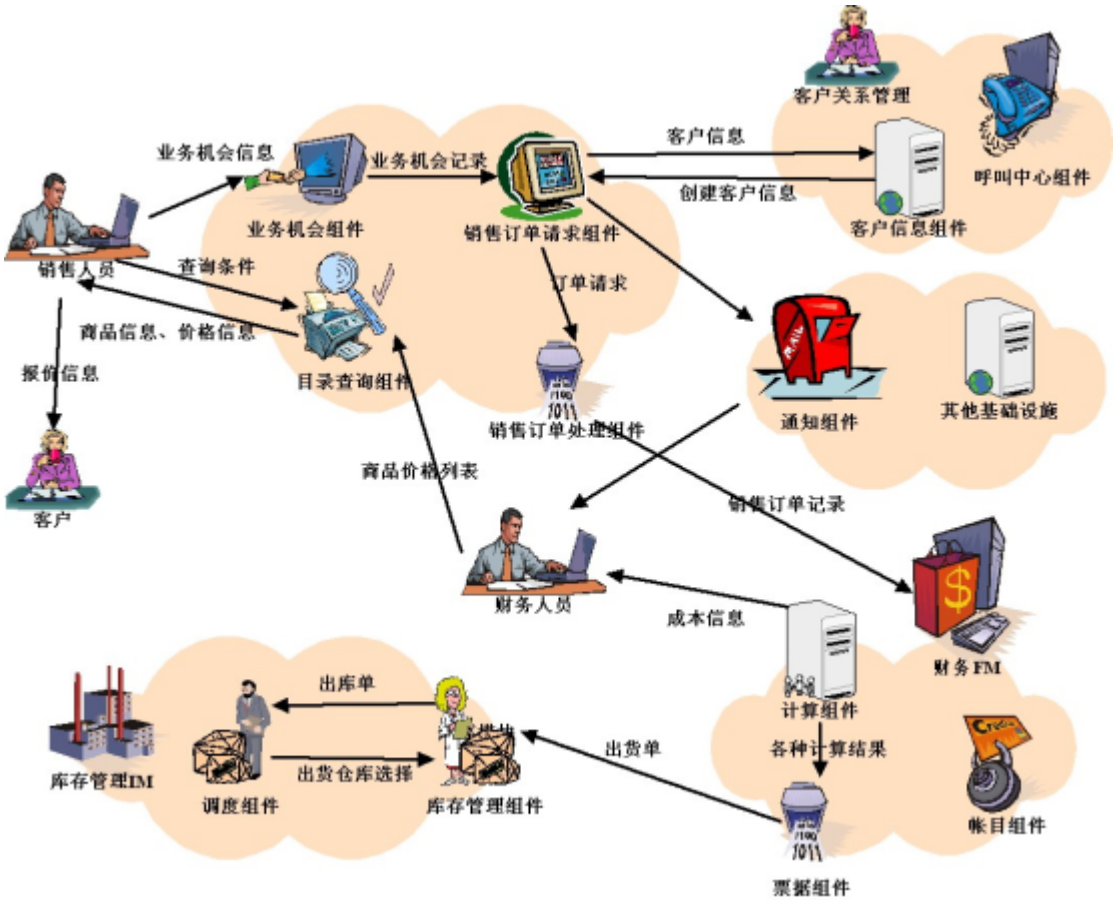
4. 系统实现结构和数据流程

组件是系统实现的组织形式，我们下面的系统结构图能够反应出系统中各个组件的关系:



这个系统结构图反映了我们实现的系统的结构、涉及内容和设计组件分布。在这个结构图思想的指导下，我们会逐步完成组件的内容，并随之完成每个组件的对应服务。

系统的结构图从整体上体现了我们希望实现的系统，而下面我们给出的是数据流程图，这个数据流程图反应的是数据在使用过程中的路径，为开发人员理解系统架构提供更为清晰的思路。



数据流动图示

在这个图中，反映的是所有相关人员和系统的数据交流情况，我们的实现会按照这个数据流程和前面已经提到了的接口定来实施。

系统架构和数据流程是系统状态清晰的体现，在我们完成的集成系统中，肯定也需要体现出上述数据流程和系统结构。

5. 总结

这个文档陈述了我们在组件设计方面的思想，包括组件的定义、组件的接口、组件的实现方法等等内容，并给出了系统结构图和数据流程。我们希望能够通过这个文档展示我们对于组件设计方面知识的认识。

SOA 语义化

1 Web Process 简介

到目前为止，我们涉及到的流程大部分都是企业内部的，这远远没有发挥出来 SOA 和 Web Services 的优势。SOA 和以往的分布式系统最大的不同就是它全部基于开放的标准协议，如 WSDL，SOAP，UDDI 等，这就使全球互联网上分布的海量应用可以轻松地交互协作，完成特定的业务，这在以前是不可想象的。基于这些理由，业界提出了 Web Process 的概念作为 SOA 的重要组成部分。所谓 Web Process，是指为了提高组织间的互操作能力而设计的下一代工作流技术，它可以使企业更紧密地与市场、供应商、竞争对手和客户互动，提供企业级的业务活动能力。

回顾流程技术几十年的发展，可以看到它经历了从工作流，到分布式工作流，再到现在的 Web Process 的进化轨迹，如下图所示。我们有理由相信 Web Process 带来的全球业务整合能力是革命性的，将会极大地释放生产力。

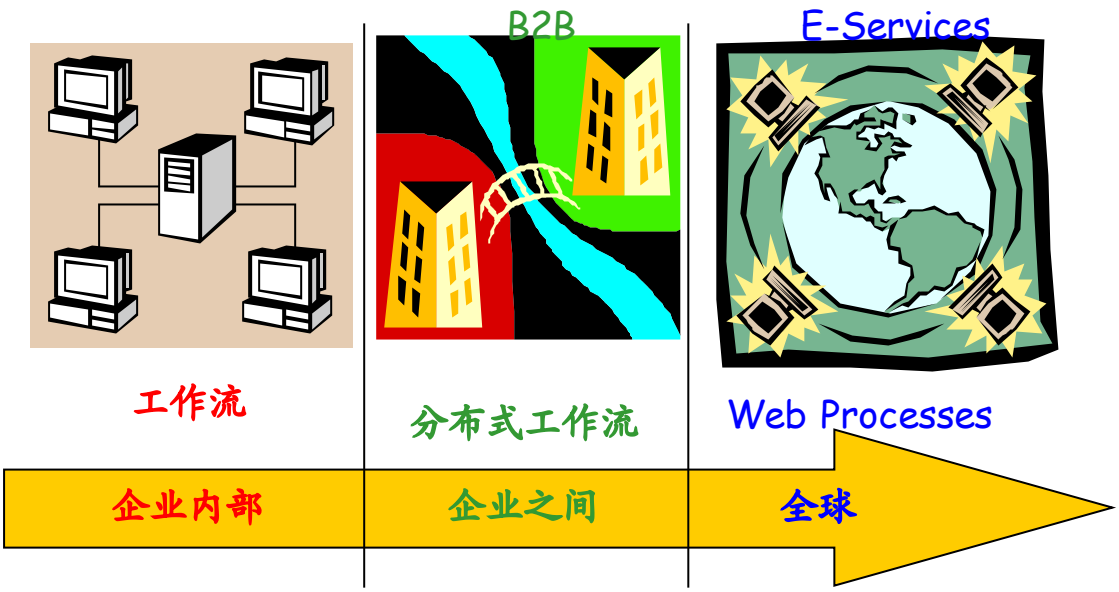


图 1 流程技术演化历史

Web Process 可以对企业内和企业间的工作流程统一建模，这个过程主要通过 BPEL4WS 建模语言组装和编排 Web Services 来完成的。Web Process 描述了怎样把众多的 Web Services 连接起来组成可靠的业务解决方案，这些 Service 既可以位于企业内部，又可以来自合作伙伴，比如供应商，物流公司，银行等。总之，Web Process 在企业中的角色就是把各种用不同技术实现的跨部门跨企业的业务应用和流程整合起来。下图是凤凰公司的销售流程，可以看到在这个过程中至少涉及到了 3 个企业。

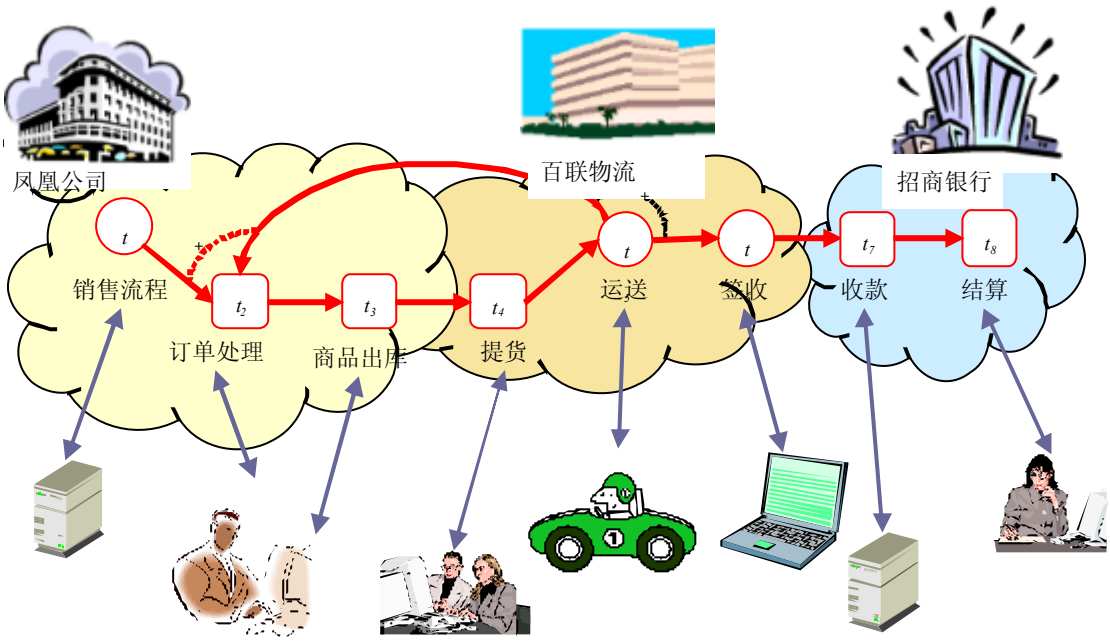


图 2 凤凰公司流程示意图

Web Process 可以用业务流程建模（BPM）的方法建立，我们把销售流程分成 8 个 Web Process，详见服务模型与设计文档中的服务发现部分。有了流程之后，就可以结合业务目标和现有系统的分析将其分解成单个的服务，正如我们在服务发现中做的那样。这个过程可以用下图表示：

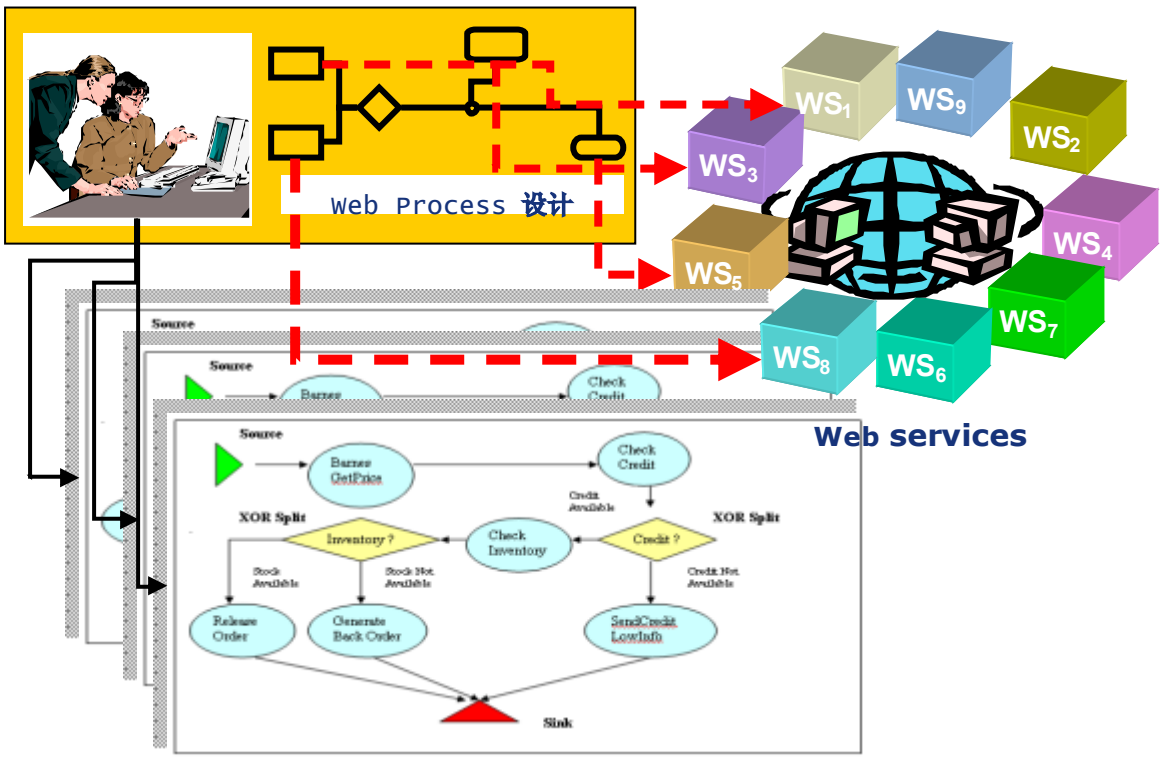


图 3 过程与服务关系示意图

2 SOA 技术面临的挑战

前景是美好的，道路是曲折的。 SOA 架构需要考虑到的不光是企业内部自闭的系统，而是全球范围内开放的，庞大的应用系统，这就不可避免地带来很大的挑战，包括:

2.1服务的异构性和自治性

这包含语法的异构，语义的异构，功能的异构。试想各个企业的业务应用接口互不相同，对某个术语的含义理解不同，甚至同一应用的功能和性能也不相同。就算上述方面都相同，各企业间的业务规则也不可能绝对统一。这种情况在全球开放环境下是相当普遍的，极大地影响了 B2B 和电子商务的发展。

解决这个挑战的办法就是用一种机器可以理解的方式描述服务。当前用来描述服务的 WSDL 语言只能描述接口和绑定，还缺乏描述丰富的语义的能力。这意味着两份内容完全相同的 WSDL，不同的人会作出不同的解释。

2.2 业务交互的动态本质

IBM 提出了“世界上唯一不变的就是变化”这个著名论断，SOA 也是为了适应业务变化提出来的，但是现在的技术和架构真的够用吗？当环境变化时我们真的能在第一时间作出响应吗？现在的 SOA 技术可以用 BPEL4WS 对服务进行组合，但只限于***设计时***的静态组合，而不能实现***运行时***的动态绑定、组合和演化。

为了更好地理解这一点，我们考察一个凤凰公司采购流程的真实场景：当生产车间需要订购原材料时，会到 service 注册中心寻找所有可用的供应商，凤凰公司向这些供应商发出报价请求，得到响应后，就会根据它们的服务质量选择一个供应商，并通过它的订购系统发出订单。

对这个简单的流程，我们考察一下现有 SOA 系统是怎样处理的。可以看到，选择哪个供应商的服务，是在设计时通过 BPEL4WS 指定的，是一种静态的服务绑定。当凤凰公司想更换供应商时，需要用 BPEL 重新设计流程，如下图所示:

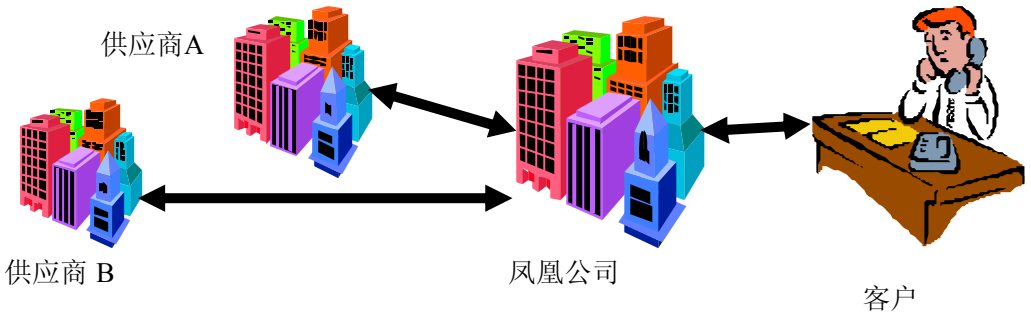


图 4 服务绑定场景之一

当然，由于不需要改动代码，这种通过 BPEL 重新编排服务的方法已经比以前进步了很多，但是还是不能满足企业快速响应变化的需要。企业需要的是动态绑定，从而在运行时动态选择最好的供应商。只要订好规则，明确什么是“最好”，就能由系统自动改变流程，如下图所示:

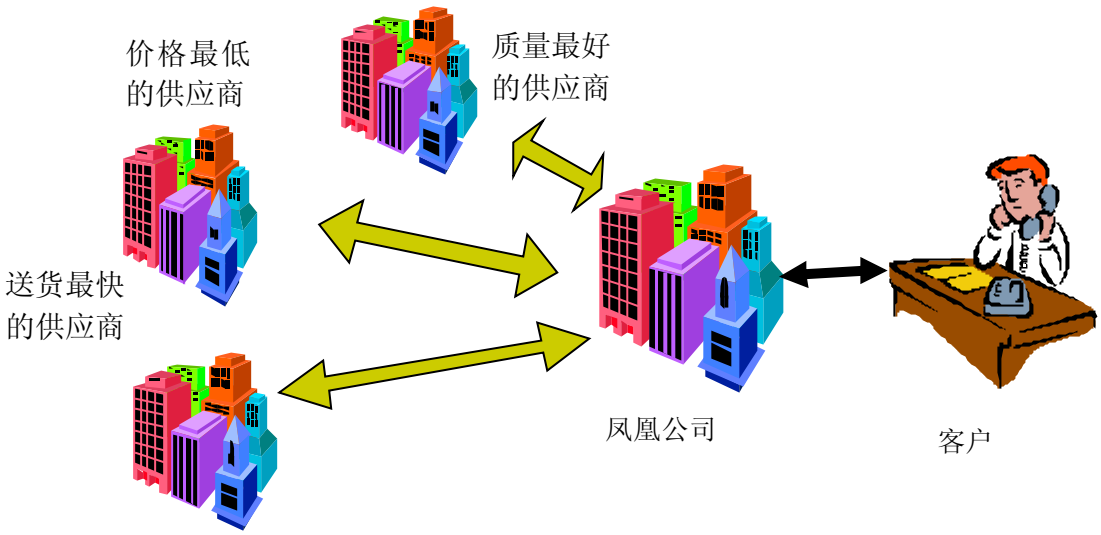


图 5 服务绑定场景之二

由以上这个小场景可以看出，SOA 还需要进一步支持相关动态特性，以更好地满足企业适应需求变化的要求。

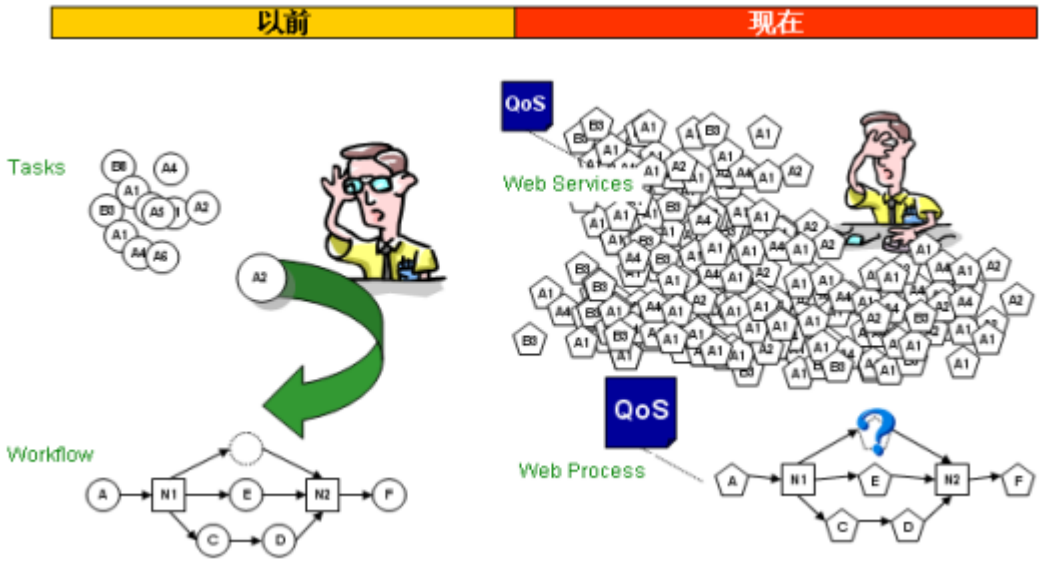


图 6 服务规模扩大带来的新问题示意图

2.3 服务的规模

当业务从企业内扩展到整个互联网时，浩如烟海的服务和数据不再是人手工能处理的了。这时候必须要有一套高效的服务自动发现，选择和组合的机制。现在 SOA 的做法是利用 UDDI 服务注册中心，但是 UDDI 离我们的期望还差得很远，UDDI 很大程度上还要依靠人工进行对关键字的选择和对结果的筛选，不符合自动化的要求。

3 语义化——SOA 的下一个方向

为了解决上述的各种挑战，学术界正在把语义网技术和 Web Services 技术结合起来，试图构建下一代的语义化 SOA 体系。下面我们先简单解释几个术语。

3.1 释义

有人说过，要使计算机变得更聪明无非有两种方式：第一种是增加机器的智能，让机器推理出数据的含义，从这个方向产生了人工智能这门影响深远的学科；第二种方式则是增加数据的智能，使数据更容易地被机器发现和处理，这就是我们要提到的语义学（Semantics）。使数据具有语义，可以大幅度地简化对机器处理的要求，可以使机器“读懂”数据，从而做出自主判断。现在的企业间数据交换主要用的是 EDI 技术，但 EDI 最大的问题是它是一种封闭的技术，难于实现和维护，而且任何一个 B2B 通讯都要单独编写相应的程序。

可见，只有语义化，使不同的术语都遵从统一的领域模型的定义，才能实现 SOA 所宣称的远景目标，使业务合作伙伴关系确立的代价最少。这样，我们需要把第一节提到的 Web Process 的描述语义化，从而演化为 Semantic Web Process。

本体论（Ontology）是语义化的基础。所谓本体，是指一组被赋予了特定含义的概念和

概念之间的联系。创建本体的目标是在某一领域建立统一的词汇表，使信息的交换去除二义性影响。

3.2 SOA 中的语义

SOA 的语义化包含 4 个方面的语义化，涉及到 SOA 生命周期中的 4 个阶段。它们分别是:

3.2.1 服务数据的语义化

所谓的数据语义化，指的是对 Web Service 输入输出的消息进行形式化或半形式化的定义。只有严格定义了语义，使互联网上千差万别的 Service 遵从标准的语境，可以极大地提高服务发现和服务互操作的能力。数据语义化可以靠上一节提到的本体(Ontology)实现，只要在 Service 的输入输出的消息中加上一些注解，使这些消息关联到已经定义好的本体上，就可以方便地实现语义化。关于注解的详细说明，请参看下面的章节。

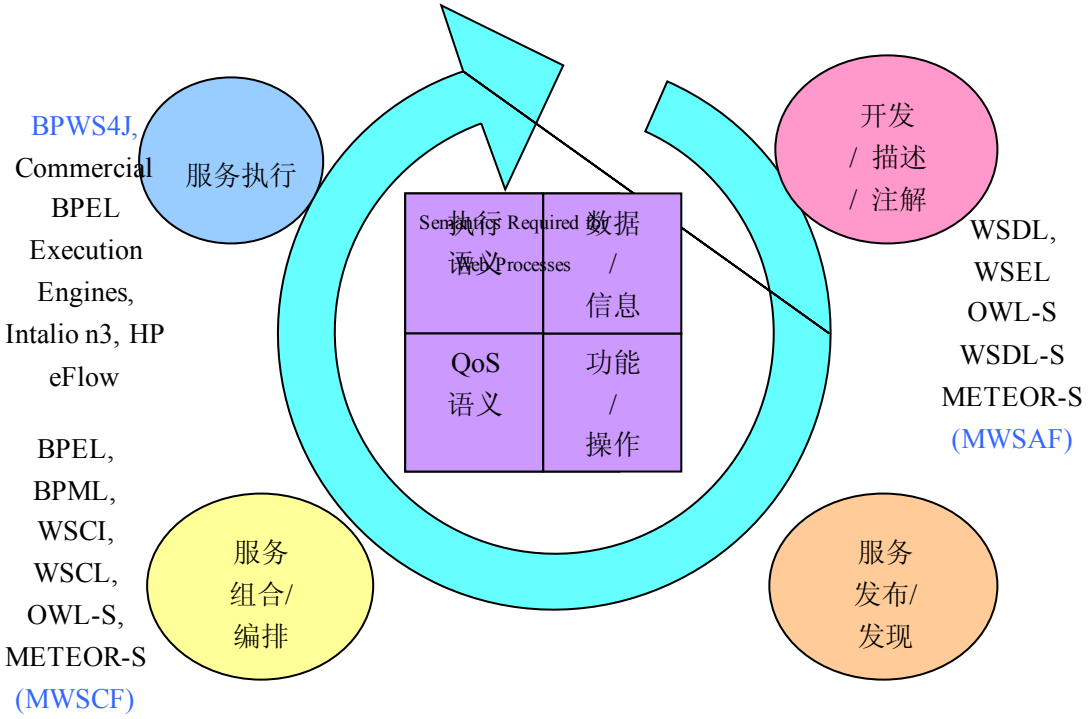


图 7 SOA 语义化示意图

3.2.2 服务功能的语义化

只有 Service 的输入输出消息语义化还不够，我们需要对 Service 的功能描述也语义化。先考察一下现在描述 Web Service 功能的手段：我们登录某个 UDDI 注册中心，利用它的分类目录(类似电话黄页)手工查找需要的服务，或者键入一些关键词，从服务的文字描述里查找；选定服务后，需要下载它的 WSDL 描述文件从而得到它的接口。

OK，我们看一下这个过程，对服务功能的描述是文字形式的，它的语义完全需要人脑来解读，且不说人脑会对相同的文字描述产生不同的理解(二义性)，就算语义理解正确，人

工处理也不符合我们自动化的要求。所以，不光需要人能理解服务描述，机器也应该能理解。

为了使机器理解服务的功能和执行的操作，促进服务选择和组合的自动化，我们需要扩展现有的 UDDI 技术，使其语义化。

3.2.3 服务执行的语义化

如果说上一节提到的服务功能描述的语义化是为了更好的进行服务选择和组合，那么这一节提出的服务执行的语义化，则是为了使服务执行的流程(Web Process)自动化。现在用来描述流程的 BPEL 本质上是一种静态的，设计时的绑定，正如我们在 2.2 节提到的场景那样，它还不能满足企业快速响应变化的需求。只有形式化地表达出服务如何组合成一个流程，自动对流程进行分析，验证，模拟，当外界发生变化时，自动调整流程进行响应，才能建立起运行时的绑定，真正达到随需应变的境界。

3.2.4 服务质量(QoS)的语义化

QoS 在 SOA 中扮演着重要的角色，比如 2.2 节场景里的供应商选择，就是基于各供应商的服务质量。但问题是各个企业对服务质量的度量方式，衡量标准等存在严重的不一致。比如供应商 A 和 B 的服务响应时间单位不一样，或者故障率的统计方法不一样，就会对我们基于 QoS 的服务选择无法进行。

幸运的是，现在已经有一些 Web Service QoS 的标准模型和描述它们的本体(Ontology)，我们只要应用这些本体，就可以实现 QoS 的语义化，使机器自动处理服务的评价，监控和选择。

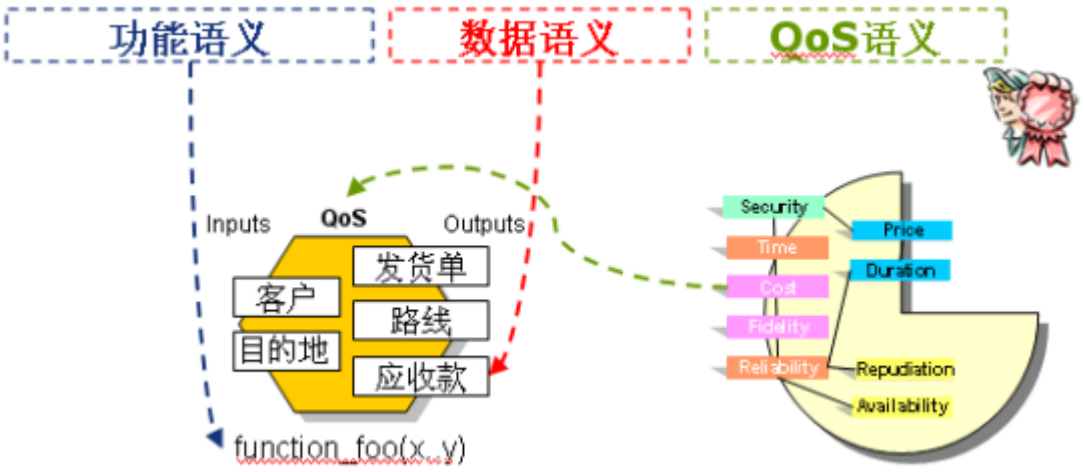


图 7 服务语义化的三方面要求

3.3语义化的三个层次

如右图所示，SOA 的语义化主要分为三个层次：

1. 最底层是**服务描述的语义化**，上一节提到的服务数据，服务功能和 QoS 都是对服务的描述，都属于这一层。它的功能是使服务消费者对服务有一个机器可处理的，无二义性的理解。这主要是对现有的 WSDL 接口文件添加语义注释实现的。(WSDL-S)
2. 中间层是**发布和发现的语义化**，这一层主要解决的是如何使机器自动快速处理服务的发布，以及如何在海量数据中发现需要的服务。

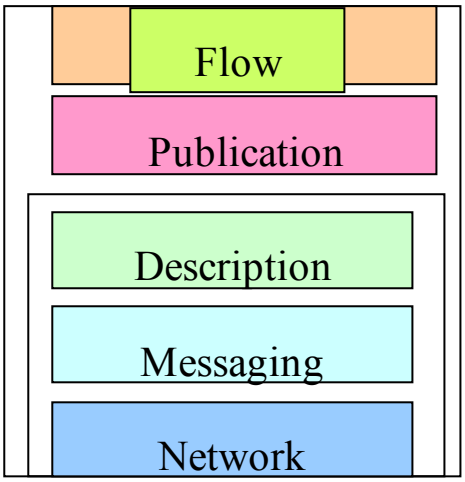


图 8 服务语义化的三个层次

1. 最上面一层就是**流程的语义化**，实现了服务的组合，分析，验证和执行，这也是我们实现流程自动化的必由之路。最近，OWL-S 提出了仲裁器体系结构进行控制流和数据流的动态重组。实际上这一层次的实现是最为困难的，涉及到一些形式化的数学模型，比如流程代数，同步机制，状态机，Petri 网等等，是需要进一步研究的领域。

下面我们就逐层考察它们的语义化是如何实现的。

4 服务描述层的语义

目前 WSDL 是描述 Web Services 的标准，它是一个 XML 文件，包括对服务接口的抽象描述和对服务绑定的具体规定。目前这种描述能力还比较弱，为了更好地发现，组合和编排服务，我们需要对接口描述做一些扩充。

一个比较好的方案是用一些语义元数据 (Semantic meta-data)对 WSDL 中的接口进行注释，使其关联到相关的本体(Ontology)上。包括服务的输入，输出，功能描述都可以映射到对应的数据本体，功能本体，操作本体。在这里我们用到了 XML schema，消除了用 XML 消息表示的 WSDL 名词和本体概念之间的二义性。

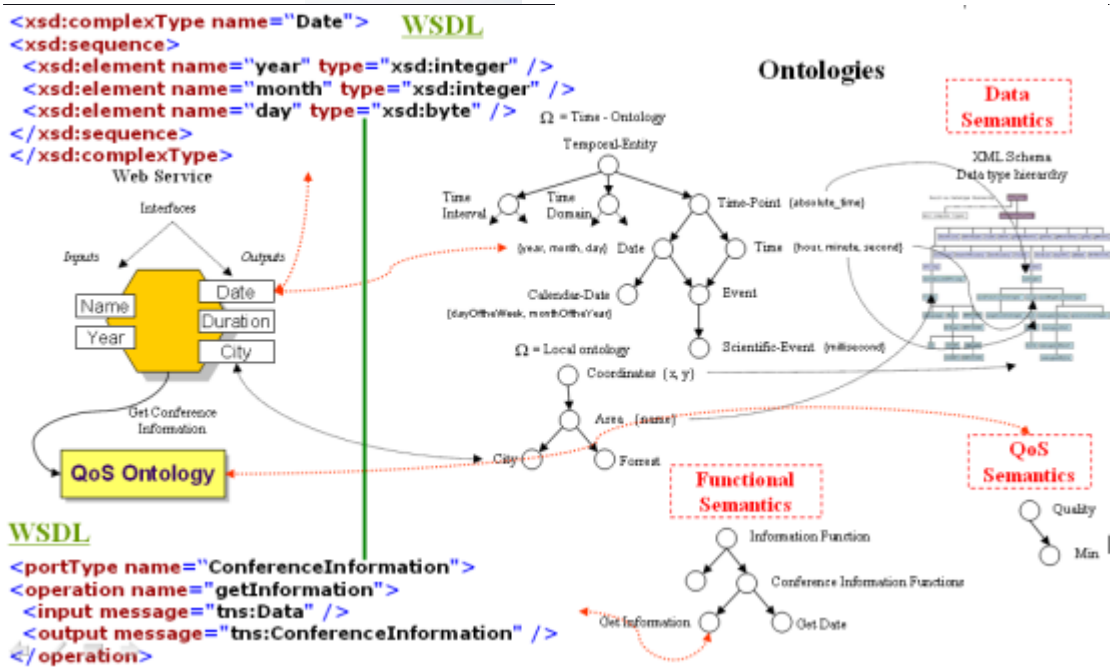


图 9 服务语义化的实现

如上图所示，我们可以把左边的 WSDL 文件和右边的本体对应起来。这方面的先驱是 OWL-S，它的前身是美国国防部的 DAML-S，它用 OWL 语言(一种标记语言，脱胎于 RDF)定义了一套 Web Services 的标准本体。

5 发布和发现层的语义

5.1 UDDI 扩展——面向语义的服务搜索引擎

目前的服务发现和发布标准是UDDI。作为服务的注册中心和目录库，它就像Web Services 的 DNS 一样，扮演着非常重要的作用。

但是现在 UDDI 的发现机制是基于关键词和属性相关的搜索，这不符合我们的期望。我们希望搜索引擎可以更好地理解服务的内容，从而做出更精确的搜索。在上一节我们已经向 WSDL 接口文件里添加了语义原数据作为附加信息，引入了领域本体，如果把这些信息同时发布到经过扩展的 UDDI 服务器，就可以使搜索引擎事先不需要人工干预的自动服务发现，从而极大促进了多个 Web Services 之间的集成。

对调用服务的需求，包括它的功能，输入输出，服务质量，前置条件后置条件和效果，都可以用一个模板来描述，这个模板可以用本体来构造。把模板和 UDDI 中的所有注册项进行匹配，就可以找出最符合要求的那个服务。如下图所示:

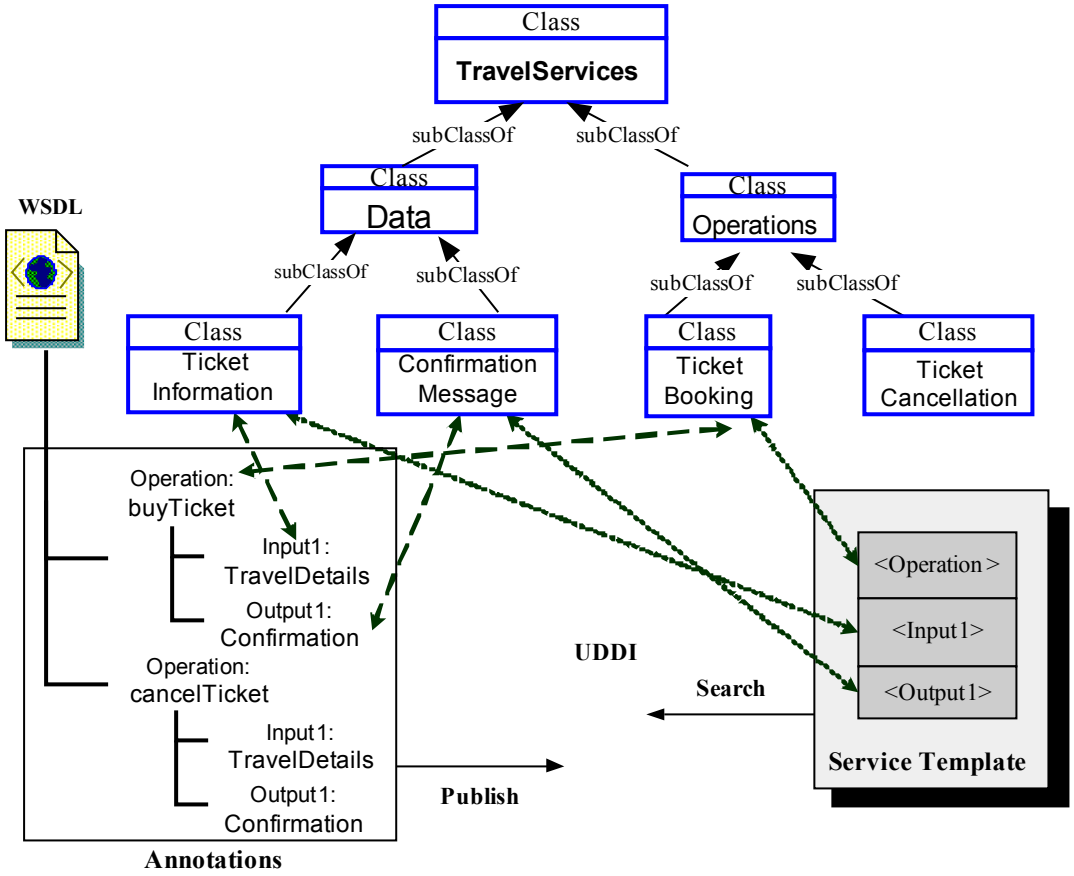


图 10 服务的语义表示

5.2 匹配算法

那么到底哪个服务才是最符合要求的服务呢？在这么多不同的服务中如何取舍，就成了匹配算法要解决的问题。我们需要定义一种查询语言，提交描述需求的服务模板(ST)，在众多的服务对象(SO)中应用匹配算法，找到相似度最高的服务。整个过程如下图所示：

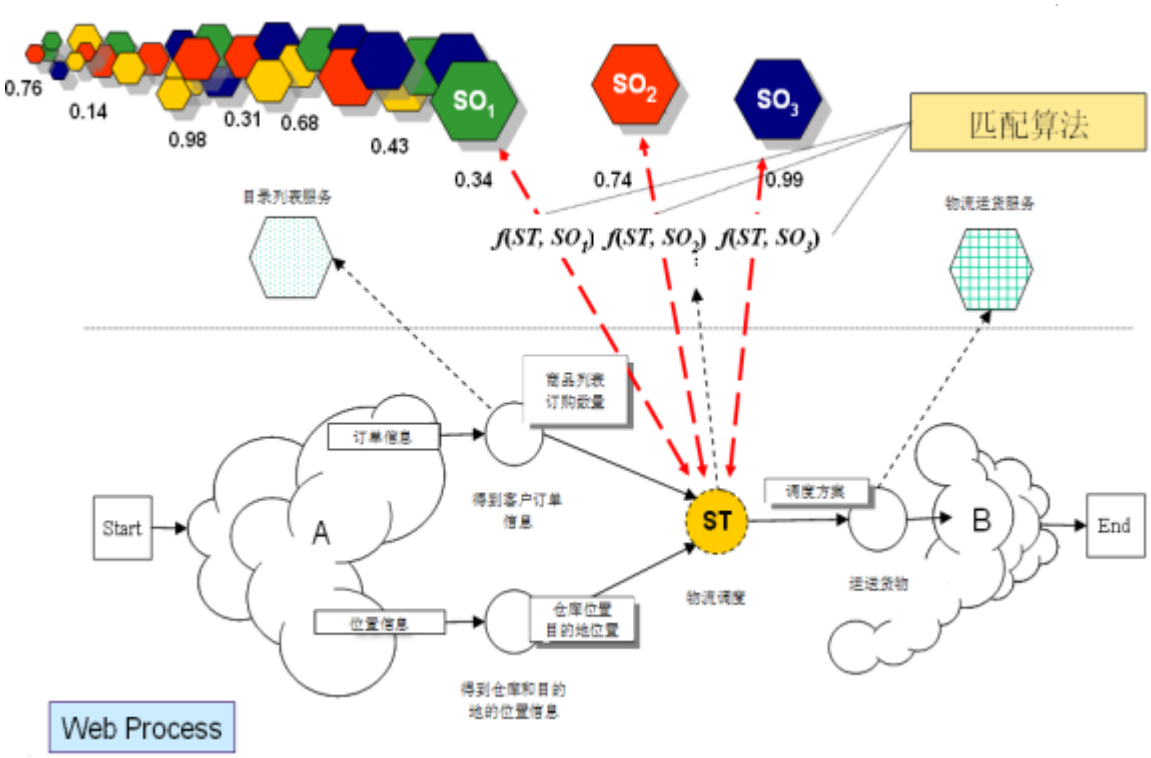


图 11 服务调度与匹配过程示意图

服务的相似度包括服务表示句法的相似，QoS 的相似和功能/数据的相似。句法的相似可以用一些语言学的方法解决，比如用 Word-Net，可以在所有的词汇间建立联系，把相似的词汇聚集在一起。仅仅分析句法的相似度是远远不够的，还要评估 QoS 的匹配程度，这需要用到 QoS 的标准本体。当然最重要的是功能和数据的相似度，这是我们评估的最主要因素，服务能不能集成到一起，主要看这一步的评估结果。服务的相似度如下图所示，具体的匹配算法有很多，我们选择了其中一种参考实现，详见附录。

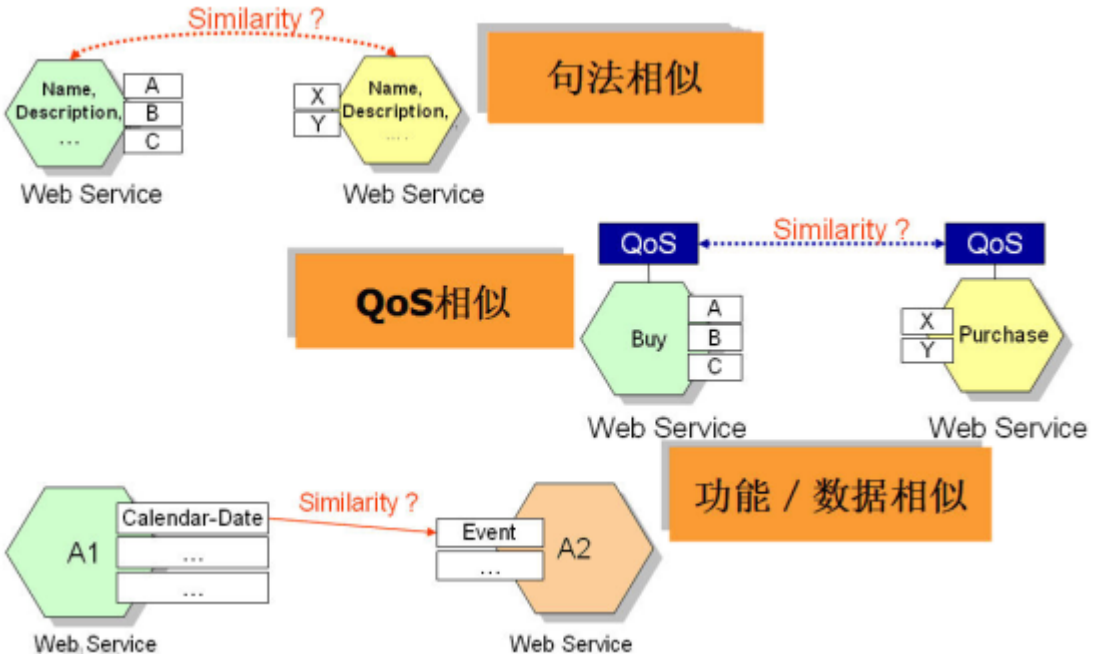


图 12 服务相似度的示意图

6 流程组合的语义

所谓组合，就是把现存的 web 服务和其他组件组合连接起来，形成新的流程的过程。组合分为两种，静态和动态：静态组合是指设计时就决定了组合方式；而动态组合是指在运行时才决定组合方式。

当需要的服务被发现后，并不能直接调用，我们还需要用一种机制消除结构和语义上的差异。这样做的原因是在第一步发现的服务往往和现有流程的其它服务是异构的。注意我们现在讨论的是整个互联网上的形形色色的服务，而不是组织内部经过良好接口设计的服务，不能期望这些服务一定符合我们的要求。

异构包括结构上的异构和语义上的异构。结构性异构指的是 Web 服务会使用不同的数据结构(或类层次结构)来定义接口的参数；语义性异构是指必须考虑到接口参数涉及到术语的真正含义，只有这样 Web 服务间交换的数据才可以被真正的理解。

这方面的成果可以参考 Cardoso and Sheth 的工作（Compute the optimal matching using semantic information, 2002）。除此之外，也有一些工具对语义映射提供了支持，比如 XMLSpy SemanticWorks 配合 MapForce，如下图所示：

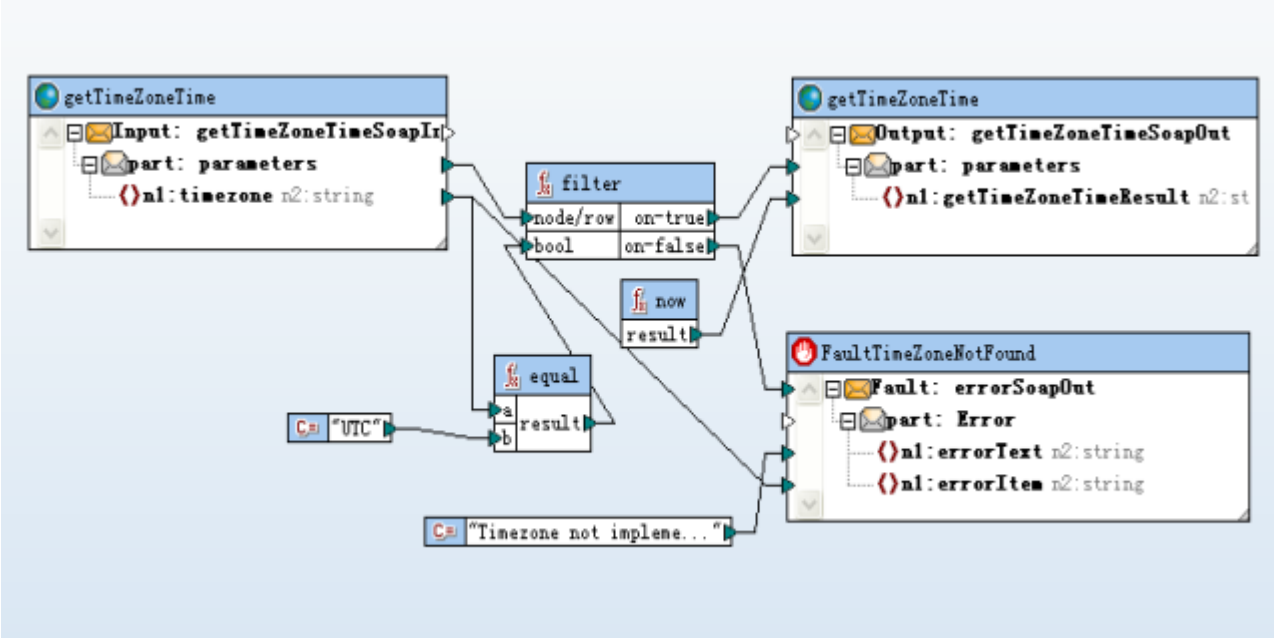


图 13 服务语义映射的示意图

7 服务质量管理

由于 Web 服务的自治性，在设计时就确定服务的质量度量是不现实的，但是当我们组合流程时，服务质量度量又是不可缺少的，所以必须在服务调用时确定质量相关的属性。

现在 QoS 管理方面还存在一些问题，比如规约的问题，QoS 模型应该包括哪些尺度？算法问题，用什么算法计算分析和预测服务的 QoS？监控问题，用什么工具随时监控服务质量的状态？还有控制问题，当某个服务的质量不满足标准时，用什么机制进行响应？

毫无疑义，解决方案还是要深入研究，建立质量相关的精确的语义表示。幸运的是现在已经有一些学者提出了用于 Web Services QoS 的本体，把原本散乱的 QoS 属性规范化，如下图所示：

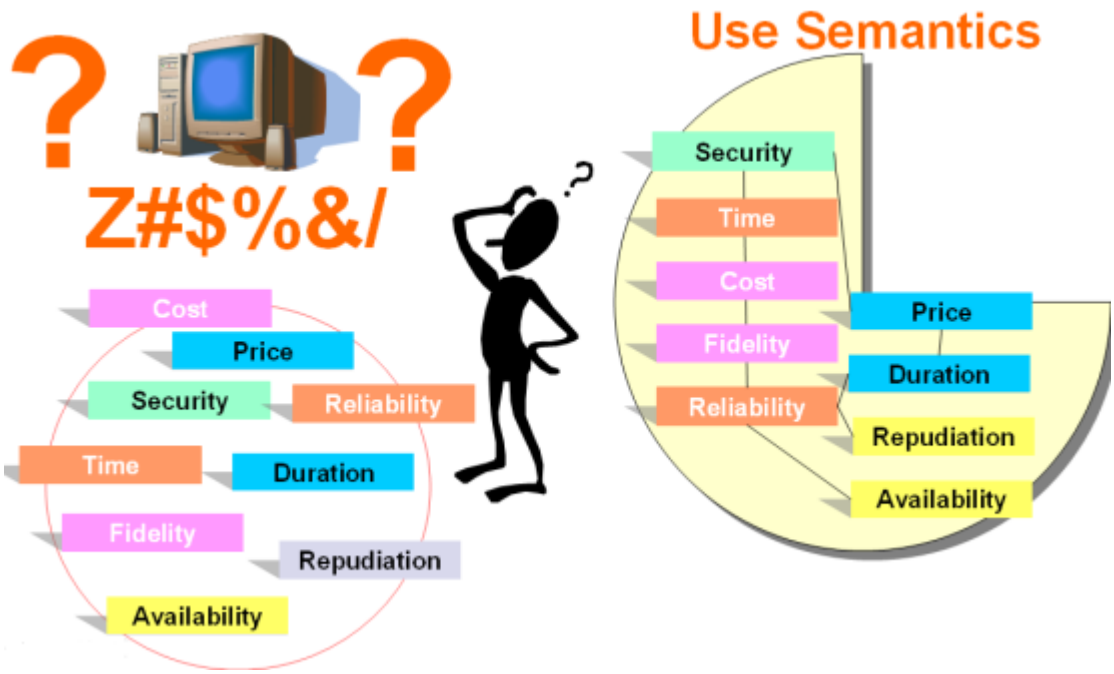


图 14 服务语义映射的困境

QoS 的计算和管理是一件比较复杂的工作。现在可用的方法有线性规划，模拟， Petri 网分析，图论和关键路径法等，目前学术界正在寻找更适合 SOA 环境下的 QoS 计算方法。作为例子，下面引用一个 Glen Dobson 提出来的 QoSOnt 结构图，具体参见附录。

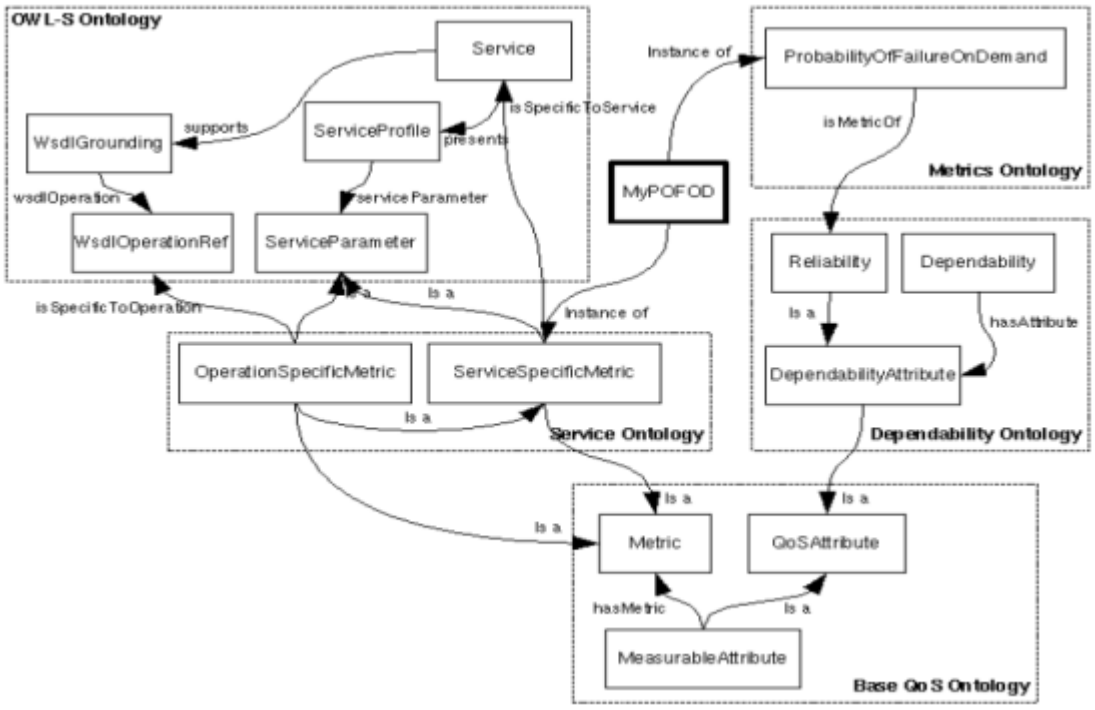


图 15 QoSOnt 结构图

附录

参考文献

[Resources] http://lsdis.cs.uga.edu/lib/presentations/SWSP-tutorial-resource.htm

[Kreger] http://www-3.ibm.com/software/solutions/webservices/pdf/WSCA.pdf

[Sivashanmugam et al.] Adding Semantics to Web Services Standards

[Sivashanmugam et al.] Framework for Semantic Web Process Composition

[Verma et al.] MWSDI: A Scalable Infrastructure of Registries for Semantic Publication and Discovery of Web Services

[Chandrasekaran et al.] Performance Analysis and Simulation of Composite Web Services

[Cardoso et al.] Modeling Quality of Service for Workflows and Web Service Processes

[Silver et al.] Modeling and Simulation of Quality of Service for Composition of Web Services

[Paolucci et al.] Importing Semantic Web in UDDI

[UDDI] http://uddi.org

[DAML]http://www.daml.org/services/

信息智能服务社区

以上我们详细阐述了 SOA 语义化的必要性和重要性。我们知道，一旦跨越了组织边界，服务的发现，获取和管理就成了影响 SOA 操作效率的主要因素。但是现在的服务支撑机制还大多使用基于传统的面向对象的模型，这严重影响了服务发现和选择的效率。因此，学术界与工业界一致认为迫切需要实现服务发布和发现的智能化和自动化的支撑机制。幸运的是，网上已经出现了若干信息智能服务社区（Information Intelligence Service Community），它本质上是一个信息处理服务的创建和运营平台，任何一个注册的机构都可以把自己的信息处理服务发布到这个平台上，前提是必须遵守平台规定的发布标准。经社区委员会评审通过后可以由这个平台来管理和运行所发布的服务。任何第三方如果要使用此项服务，需要向服务提供方交纳一定的服务费，例如，采用订阅的方式，社区委员会从中按比例收取佣金。注册的机构也可以使用平台上已有的信息处理服务来创建自己的增值服务，包括将已有的服务组合成新的特色服务，新的服务也可以发布到平台上去。

我们认为信息智能服务社区的构建主要涉及以下方面：信息智能服务构件的体系结构；社区管理及运行机制。 下面我们从这两方面进一步展开讨论：

1、信息智能服务社区需求分析

我们知道，Web 服务的最大优点之一即是它能够跨越组织和系统的边界，实现动态的服务发现，组合和互操作。但是现在的服务模型大多基于面向对象的计算模型，对象模型固有的被动性和数据为中心的特点在很大程度上制约了服务以下性质的展现：自主性、协同性和演化性。因此，学术界与工业界一致认为迫切需要一种能够反映和支持上述特性的计算模型来支持 Web 服务的操作运行。它不能像传统的信息系统功能模块那样被动的等待来自用户以及其他应用程序的调用，而应该主动的感知环境需要，并为满足该需要建立并提供最优的解决方案。

要实现这个目标，主要有以下几方面的因素需要考虑。首先，要将服务以及服务请求发布出去；其次，服务发布者提供服务是为了盈利，所以合理的付费机制也是必须考虑的，这包括决定服务以什么样的方式，什么样的价格提供给消费者；再次，要对所有候选服务进行排序，使请求者能获得最优的服务。最后，由于这是一个开放的环境，如何应对海量服务的考验，合理分配有限的计算资源也是需要解决的问题。

面向主体的技术源于分布式人工智能领域，现已融入各主流计算机研究方向之中。面向主体的风格自提出以来一直受到广泛关注，各种类型的软件主体大量用于信息处理、办公自动化、交通管理、私人助手等领域。网络本身所固有的开放、复杂、异构、动态、分布等特性，为主体技术的充分发展创造了有利条件，

同时也提出了有力挑战。基于这些考虑，我们认为面向智能主体的分布式计算模型是解决上述问题的最佳选择之一。

我们将每个服务相关实体看作一个智能主体，若干个智能主体的集合即构成一个虚拟的信息智能服务社区，即每个社区都是由若干个相互协作和通信的服务主体组成。服务模型的分析过程包括两个步骤：服务主体模型的建立和服务交互模型的建立。服务主体模型定义主体的能力和要求，每个角色拥有一定的服务能力的同时有一定待满足的服务要求。权利是通过对服务角色能够使用的资源和不能使用的资源来定义的。职责是通过对服务角色必须满足的功能来定义的，职责分为两类：一是功能性职责，一是质量性职责。功能性职责是指主体一定要做的事，质量性职责通过一组约束表达表达式来表达，即主体要保证该约束所定义的条件得到满足。 交互模型定义服务角色之间的通信协议。每种协议定义一种主体间的交互关系。协议通常指出通信的目的，发起者，接收者，输入信息，输出信息。将交互模型与角色模型相结合建立服务社区的行为规范。

例如，一些学者提出了“**服务集市**”的概念，试图针对 Web 服务引入真实世界中经济学的市场机制。我们知道市场可以通过价格体系调节供需之间的平衡，从而有效分配资源。既然 Web 服务也是一种资源，那么引入市场机制就是顺理成章的了。但是传统的市场机制比较低效，不太适用于迅速变化的 Web 服务交易，所以我们引入了基于语义匹配的**竞拍机制。**假定上面提到的信息智能服务社区是采用服务集市的业务模式，则服务集市构成 SOA 应用中的一个重要的基础设施之一，我们以服务集市为例分析智能服务社区的需求。

以下是这个平台的几点业务需求:

- ✧ 应该支持异构服务之间的交易;
- ✧ 应该允许多个服务主体（分别充当买方和卖方）同时进行交易;
- ✧ 应该允许服务动态的加入或退出社区;
- ✧ 应该确保好的服务被选择并得到执行;
- ✧ 平台应该处理服务间的依赖性，即可以把一组相关的服务打包进行交易;
- ✧ 由于不同的服务有不同的质量（QoS）和策略规范（Policy），支持服务的这些属性也是必需的;
- ✧ 另外，还需要一个合理的计价计费机制。

明确需求之后，我们就开始服务模型的设计。

2、面向主体的服务模型(Agent-Oriented Service Model，AOSM)

基于上述对智能服务社区需求的分析，我们决定采用面向主体的体系结构来搭建服务的创建和运营平台。该平台的核心思想是将每个服务相关实体看作一个

智能主体，由该主体负责处理与服务质量，服务价格等语义相关的任务。主体之间相互通信交互，其主旨是形成一个服务供应方与消费者的生态系统，从而支持服务社区的动态性，连续性，演化性和适应性。我们的主体模型具有以下特点：主体具有决策能力，主体运用本体方法表示共享知识。我们的服务模型最主要的作用在于支持优质服务的选择。

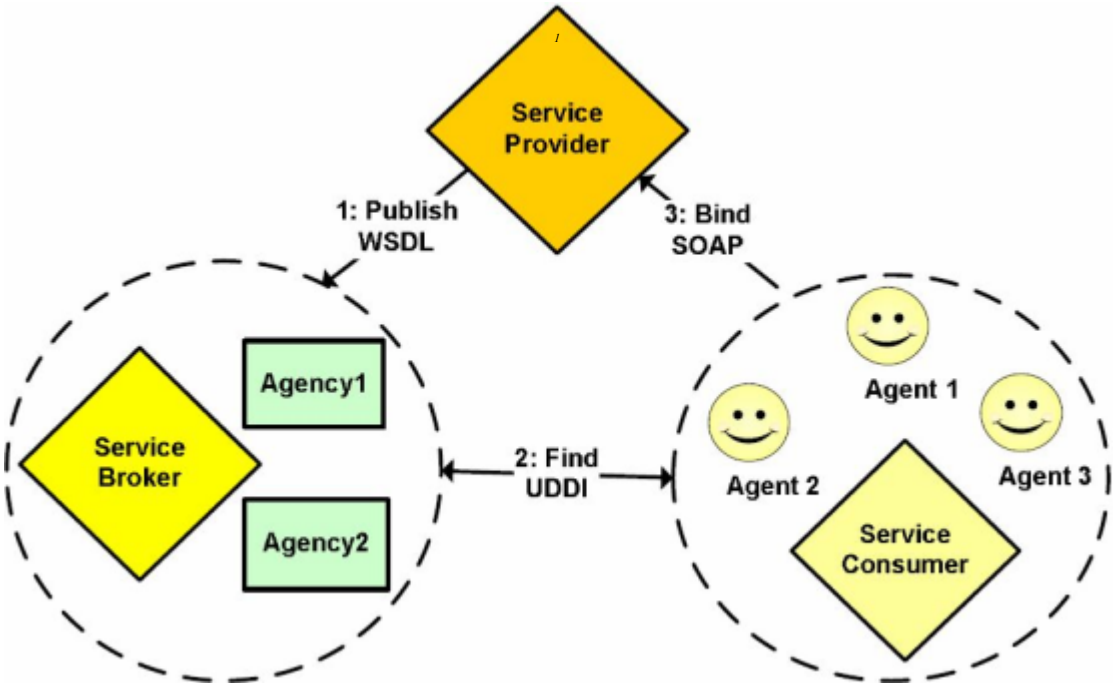


图 1 面向主体的服务应用示意图

在传统的 SOA 模型中，服务供应商将服务发布给服务注册中心。在我们的模型中，服务将发布给各个服务社区代理（Agency），服务消费者向社区代理发出查询请求，之后根据返回的结果进行选择，并与最终选定的服务供应商签约。本文采用的面向主体的结构的应用如图 1 所示。

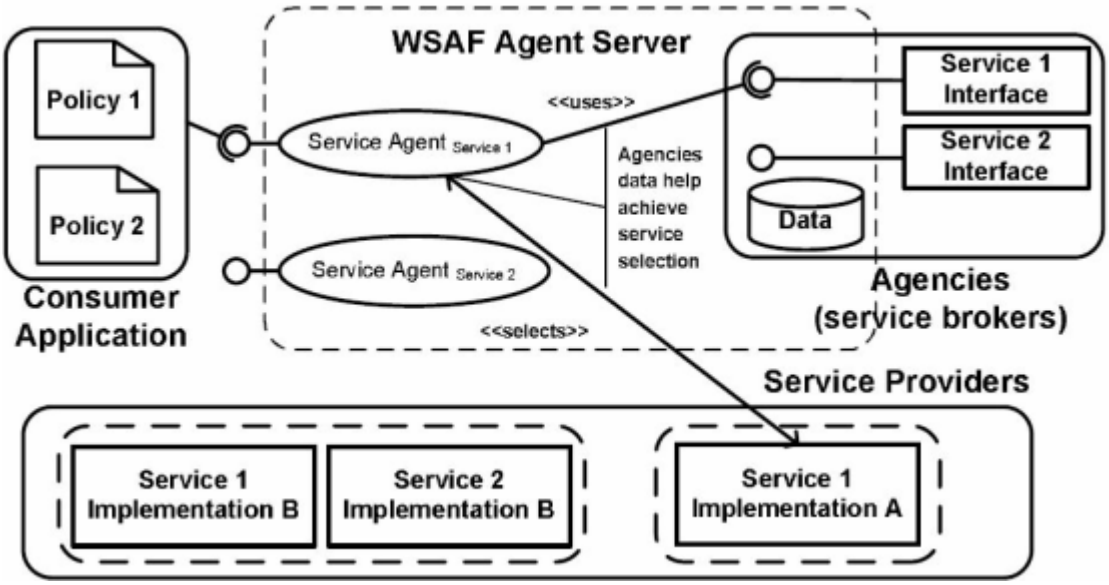


图 2 面向主体的服务体系结构图

图 2 给出本文采用的面向主体的服务体系结构图。从图中可以看出，不同服务供应商为同一服务接口（Service 1 Interface）可以采用不同的实现手段（Service 1 Implementatio A， Service 1 Implementation B）。同一供应商可以提供多个不同的服务的实现(Service 1 Implementation B，Service 2 Implementation B)。凤凰公司的系统在上图中扮演服务消费者的角色，它可以通过智能主体服务器对社区中的各种服务做出明智 的选择。本文中的面向主体的服务体系结构参照 [1]完成。下面对体系结构中所涉及的 4 类核心模块逐一进行介绍:

✧ 服务供应商(Service Providers)：社区中的每个服务供应商运用 WSDL 描述其服务，其中既包括对接口的抽象描述，又包括与具体实现相关的绑定信息的描述。

✧ 服务中间代理(Service Brokers)：服务社区的正常工作需要中间代理的支持才能进行。最主要的中间代理就是服务注册中心，它将服务信息和相关数据分类保存，帮助服务消费者找到想要的服务。在我们的系统中，服务中间代理的工作主要由智能主体协助完成。

✧ 主体服务器（Agent Server）：智能主体的宿主和管理系统。服务信息，价格信息，服务策略信息，本体描述，配置文件等均保存在此。

✧ 应用程序（Consumer applications）：服务以及智能主体的使用者，消费者。在本项目背景下，是凤凰公司的系统。

以下我们给出上述面向主体服务模型的典型交互控制流程:

✧ 步骤一：初始化主体服务器，配置相关服务中间代理;

✧ 步骤二:服务供应商向中间代理注册其服务，主要是用 WSDL 格式定义 URI，服务领域，及性能质量公告。为每个服务建立一个配套的智能主体。

✧ 步骤三：作为服务消费者的应用程序在本地创建一个智能主体的服务代理，并将选择控制策略传达给该本地代理。该本地代理成为消费应用程序的组成部分之一，专门负责与所请求服务有关的通信交互活动。

✧ 步骤四：服务器端的智能主体运用相关的策略信息及配置信息，执行关于服务质量的查询活动，为每个消费者全面搜集服务以及供应方的相关情况和信息，并在服务请求者和相关的中间代理建立联系。

✧ 步骤五：消费方调用智能主体的服务操作。智能主体跟踪和监控服务执行情况，并将获取的信息提交给相应的中间代理。

3、主体服务模型的本体描述

下面介绍我们的服务本体模型(节选自本研究小组学术论文[2])。这个模型以智能主体代表的服务各方的服务能力，待服务请求为基础建立。我们给出基本概念的定义和相关管理控制规则。下一节将给出几个小场景作为示范。这个模型的

基本假设是服务背后的每个组织或个人都是要通过服务的形式来运用自己的能力，并满足一定的需要，无论是获得物质利益，还是互换服务，还是获得社会价值。

3.1 基于策略能力的服务本体:

A Service Ontology Based on strategic capability

In this section, we will first give precise definitions on the terms in our service ontology, followed by the rules for reasoning.

✧ **主体，服务，与质量属性集合的定义。**

Definition 1. $A = \{a_1, ..., a_n\}$ *is a set of Actors*. $S = \{s_1, ..., s_n\}$ is a set of *Services*. $Q = \{q_1, ..., q_n\}$ is a set of *Quality attributes*.

✧ **服务分解与组合关系的定义。**

Definition 2. $ME = S \rightarrow \boldsymbol{P}\, S$, is a set of *means-ends* relationships. $DC = S \rightarrow \boldsymbol{P}\, S$, is a set of *decomposition* relationship s.

✧ **服务质量函数的定义。**

Definition 3. $f = A \times Q \times S \rightarrow Int$ is a set of Quality of Service function. $f(a_i, q_j, s_k)$, in which $a_i \in A$ $q_j \in Q$, $s_k \in S$, describes the value of a quality attribute q_j of a service s_k provided or required by actor a_i.

✧ **服务请求集合的定义。**

Definition 4. For each actor $a \in A$, there is an $FR \subseteq A \times S$ representing the set of *Functional Services Required* by *a*. There is also $NFR \subseteq A \times Q \times S \times Int$, representing the set of *Non-Functional Services Required* by *a*.

✧ **服务能力集合的定义。**

Definition 5. For each actor $a \in A$, there is a $FC \subseteq A \times S$ representing the set of *Capable* Functional services. There is also $NFC \subseteq A \times Q \times S \times Int$, representing the set of *Non-Functional Services Can be provided* by *a*.

✧ **服务知识(信息)集合的定义。**

Definition 6. For each actor $a \in A$, there is a $K \subseteq A \times FC \cup NFC \cup FR \cup NFR$ representing the set of *Knowledge* about services capabilities and requirements.

✧ **服务操作原语的定义。**

Definition 7. O = $\{o_1, \ldots, o_n\}$ is a set of applicable *Operations* to a service situation $SC = <A, R, C, K>$. There are following basic types of operation constructs: ***delegate***, ***tell***, and ***perform***.

1. ***delegate*** (x, y, z), represents that there is an inter-actor *delegation*, where $x, z \in A$, $y \in S$. **服务委托操作**
2. ***tell*** (x, y, z), represents an inter-actor *communication*, where $x, z \in A$, $y \in FR \cup NFR \cup FC \cup NFC$. **信息交换操作**
3. ***perform*** $_x\, y$, represents a *service delivery,* where $x \in A$, $y \in S$. **服务执行操作**

A world of services is an open environment, in which each of the above sets can be updated dynamically. In other words, actors will come into and get out from the environment. New request will be initiated or removed by actors; new capabilities will be added into or removed by the actors. In such a highly dynamic and distributed environment, automated service discovery, service agreement formation, and service selection need to be manipulated by certain machine processable rules and policies. Below, we define some of the rules that can be applied under a service situation: $sc_i = <A, R, C, K>$.

Rule 2.1: Service Delivery Rule **服务履行规则**

If an actor *a* is capable of providing a service *s*, and it also has the needs of performing the service, it can perform the service.

$$\textbf{\textit{Actor}}\,(a) \wedge \textbf{\textit{Service}}(s) \wedge \textbf{\textit{Can}}_{\,a}\, s \wedge \textbf{\textit{Requires}}_{\,a}\, s \Rightarrow \textbf{\textit{perform}}_{\,a}\, s$$

Rule 2.2: Service Composition/Transformation Rule **服务组合与演化规则**

If an actor *a* is capable of providing a set of services $\{s_1 \ldots, s_n\}$, and it also has knowledge on how to compose or transform it into other more complicated service s_0, then it will be able to provide the transformed or composite service s_0.

(1) (***OR composition, Means-ends***)

$$\textbf{\textit{Actor}}\,(a) \wedge \textbf{\textit{Service}}\,(s_0) \wedge \ldots \wedge \textbf{\textit{Service}}\,(s_n) \wedge \textbf{\textit{Know}}_{\,a}\,(\textbf{\textit{means-ends}}\,(s_0, \{s_1, \ldots, s_n\})) \wedge \textbf{\textit{Can}}_{\,a}\, s_{j\,(1 \le j \le n)} \Rightarrow \textbf{\textit{Can}}_{\,a} s_0.$$

(2) (***AND composition, Decomposes-into***)

$$\textbf{\textit{Actor}}\,(a) \wedge \textbf{\textit{Service}}\,(s_0) \wedge \ldots \wedge \textbf{\textit{Service}}\,(s_n) \wedge \textbf{\textit{Know}}_{\,a}\,(\textbf{\textit{decomposes-into}}\,(s_0, \{s_1 \ldots, s_n\})) \wedge \textbf{\textit{Can}}_{\,a}\, s_1 \wedge \ldots \wedge \textbf{\textit{Can}}_{\,a}\, s_n \Rightarrow \textbf{\textit{Can}}_{\,a}\, s_0.$$

Rule 2.3: Request Decomposition/Transformation Rule **请求分解与变形规则**

If an actor *a* requires a services *s*, and it also has knowledge on how to decompose or transform it into other more concrete services $\{s_1 \ldots, s_n\}$, then it can request for those the transformed or component services instead.

(1) (***OR decomposition, Means-ends***)

$Actor\ (a) \wedge Service\ (s_0) \wedge \ldots \wedge Service\ (s_n) \wedge Know_a\ (decomposes\text{-}into\ (s_0, \{s_1 \ldots, s_n\})) \wedge Requires_a\ s_0 \Rightarrow Requires_a\ s_1 \wedge \ldots \wedge Requires_a\ s_n.$

(2) (***AND decomposition, Decomposes-into***)

$Actor\ (a) \wedge Service\ (s_0) \wedge \ldots \wedge Service\ (s_n) \wedge Know_a\ (means\text{-}ends\ (s_0, \{s_1 \ldots, s_n\})) \wedge Requires_a\ s_0 \Rightarrow Requires_a\ s_1 \vee \ldots \vee Requires_a\ s_n.$

Rule 2.4: Publication Rule　服务出版公告规则

An actor *a* may inform other actors about its request, capability about a service.

(1) **Publish Request to Known Provider**

$Actor\ (a) \wedge Actor\ (b) \wedge Service\ (s) \wedge Requires_a\ s \wedge Know_a\ (Can_b\ s) \Rightarrow Tell\ (a, Requires_a\ s, b) \wedge Know_a\ (Know_b\ (Requires_a\ s)).$

An actor *a* may publish a request to a known provider with the intention of building a service agreement. A direct effect of this publication action is that the publisher believes that the receiver of the message will know about his requirement on this service.

(2) **Publish Request to an Expert on Service Transformation/Composition/Decomposition**

$Actor\ (a) \wedge Actor\ (b) \wedge Service\ (s) \wedge Service\ (s') \wedge Requires_a\ s \wedge Know_a\ (Know_b\ (s' \rightarrow s)) \Rightarrow tell\ (a, Requires_a\ s, b) \wedge Belief_a\ (Belief_b\ (Requires_a\ s)).$

An actor *a* may publish a request to a known expert, who has knowledge on service composition, decomposition, or transformation, with the intention of knowing relevant steps of fulfilling a service. A direct effect of this publication action is that the publisher believes that the receiver of the message will know about his requirement on this service.

(3) **Publish Request to Service Registry (or Other Information Intermediary)**

$Actor\ (a) \wedge Actor\ (b) \wedge Actor\ (x) \wedge Service\ (s) \wedge Service\ (s') \wedge Requires_a\ s \wedge Know_a\ (Know_b\ ((Requires_x\ s) \vee (Can_x\ s) \vee (Know_x\ s' \rightarrow s))) \Rightarrow tell\ (a, Requires_a\ s, b) \wedge Know_a\ (Know_b\ (Requires_a\ s)).$

An actor *a* may publish a request to a known information center, who might be a web services registry, or simply another actor, who has knowledge on capabilities, requests, knowledge on other unknown actors, with the intention of knowing relevant information of fulfilling a service. A direct effect of this publication action is that the

publisher believes that the receiver of the message will know about his requirement on this service.

(4) **Request Broadcasting**

$Actor(a) \wedge Service(s) \wedge Requires_a\ s \Rightarrow tell(a, Requires_a\ s, all) \wedge Know_a(Know_{all}(Requires_a\ s))$.

An actor *a* may broadcast a request with the intention of obtaining relevant information of fulfilling a service. A direct effect of this publication action is that the publisher believes that the receiver of the message will know about his requirement on this service.

(5) **Publish Service to Known Requestor**

$Actor(a) \wedge Actor(b) \wedge Service(s) \wedge Can_a\ s \wedge Know_a(Requires_b\ s) \Rightarrow tell(a, Can_a\ s, b) \wedge Know_a(Know_b(Requires_a\ s))$.

An actor *a* may publish a service to a known requestor, with the intention of building service agreement. A direct effect of this publication action is that the publisher believes that the receiver of the message will know about his capability on this service.

(6) **Publish Service to Known Expert on Service Composition/Transformation**

$Actor(a) \wedge Actor(b) \wedge Service(s) \wedge Service(s') \wedge Can_a\ s \wedge Know_a(Know_b\ s \rightarrow s') \Rightarrow tell(a, Can_a\ s, b) \wedge Know_a(Know_b(Can_a\ s))$.

An actor *a* may publish a service to a known expert, who has knowledge on service composition, decomposition, or transformation, with the intention of knowing relevant steps of building a new service based on existing ones. A direct effect of this publication action is that the publisher believes that the receiver of the message will know about his capability on this service.

(7) **Publish Service to Information Intermediary**

$Actor(a) \wedge Actor(b) \wedge Actor(x) \wedge Service(s) \wedge Service(s') \wedge Can_a\ s \wedge Know_a(Know_b((Requires_x\ s) \vee (Can_x\ s) \vee (Know_x\ s' \rightarrow s))) \Rightarrow tell(a, Can_a\ s, b) \wedge Know_a(Know_b(Can_a\ s))$.

An actor *a* may publish a service to a known information center, who might be a web services registry, or simply another actor, who has knowledge on capabilities, requests, knowledge on other unknown actors, with the intention of knowing relevant information of promoting a service. A direct effect of this publication action is that the publisher believes that the receiver of the message will know about his capability on this service.

(8) **Service Advertising**

Actor *(a)* ∧ ***Service*** *(s)* ∧ ***Can*** $_a$ *s* ⇒ ***tell*** *(a,* ***Can*** $_a$ *s,* ***all)*** ∧ ***Know*** $_a$ *(****Know*** $_b$ *(****Can*** $_{all}$ *s)).*

An actor *a* may broadcast an advertisement of a service with the intention of obtaining relevant information of promoting a service. A direct effect of this publication action is that the publisher believes that the receiver of the message will know about his capability on this service.

Rule 2.5: Knowledge Update Rule　知识与信息更新规则

∃ *x*∈ *R*∪ *C*∪ *K*∪ *B,* ***Actor*** *(a)* ∧ ***Actor*** *(b)* ∧ ***tell*** *(a, x, b)* ⇒ ***Know*** $_b$ ***Know*** $_a$ *x.*

An actor will update his Knowledge when receive a message about a requirement, a capability, a piece of knowledge, or belief. A direct effect of this action is that the receiver of the message will know about the relevant information .

Rule 2.6 Knowledge Contradiction Resolution Rule　知识冲突与矛盾消解规则

(1) No Contradiction: ***Actor*** *(a)* ∧ ***Actor*** *(b)*∧ ***Know*** $_b$ *(****Know*** $_a$ *x)* ∧ *no* ***Know*** $_b$ *not x* ⇒ ***Know*** $_b$ *x.*

(2) **Ignore:** ***Actor*** *(a)* ∧ ***Actor*** *(b)*∧ ***Know*** $_b$ *(****Know*** $_a$ *x)* ∧ ***Know*** $_b$ *not x* ⇒ *B* */ {****Know***$_b$ *x,* ***Know*** $_b$ *not x }.*

(3) Ask public opinion about a contradicting knowledge

Actor *(a)* ∧ ***Actor*** *(b)*∧ ***Know*** $_b$ *(****Know*** $_a$ *x)* ∧ ***Know*** $_b$ *not x* ⇒ ***tell*** *(b, not x, all).*

(4) Confirm with the sender about a contradicting knowledge:

Actor *(a)* ∧ ***Actor*** *(b)*∧ ***Belief*** $_b$ *(****Belief*** $_a$ *x)* ∧ ***Belief*** $_b$ *not x* ⇒ ***tell*** *(b, not x, a).*

(5) Accept the sender's knowledge although contradicting:

Actor *(a)* ∧ ***Actor*** *(b)*∧ ***Know*** $_b$ *(****Know*** $_a$ *x)* ∧ ***Know*** $_b$ *not x* ⇒ ***Know*** $_b$ *x.*

Rule 2.7: Service Agreement / Delegation Rule　服务合约与委托规则

Actor *(a)* ∧ ***Actor*** *(b)* ∧ ***Service*** *(s)* ∧ ***Requires*** $_a$ *s* ∧ ***Know*** $_a$ *(****Can*** $_b$ *s)* ∧ ***tell*** *(b, s, a)* ⇒ ***delegate*** *(a, s, b).*

A service agreement is established when an actor *a* has a requirement, and he believes that another actor *b* could provide the service, and also receives a message from *b* about his capability regarding the service. A direct effect of a service agreement is a delegation action.

Rule 2.8: Reciprocal Dependency Rule　对等服务规则

Actor *(a)* ∧ ***Actor*** *(b)* ∧ ***Service*** *(s)* ∧ ***delegate*** *(a, s, b)* ∧ ***delegate*** *(b, s', a)* ⇒ ***Requires*** $_b$ *s (for a).*

A delegation will take effect to the delegatee only if he believes that it is

reciprocal. That is, he also needs exchange-services from the requestor. In real world case, general exchange for services could be payment, social benefits, etc.

Rule 2.9: Capability Propagation Through Delegation 服务能力提升规则

Actor *(a)* ∧ ***Actor*** *(b)* ∧ ***Service*** *(s)* ∧ ***delegate*** *(a, s, b)* ∧ ***Perform*** $_b$ *s* ⇒ ***Can*** $_a$ *s* .

A delegation will take effect to the delegator, only if the delegatee performs the service provisioning procedure. That is, if a delegatee does not deliver the expected services, the fulfillment of the delegator's service request is problematic.

The reasoning procedure to be applied to a service situation *SC* = <*A, R, C, K* > is to find a sequence made up of links (*k*) applied to *SC* such that for each ***Requires***$_a$ *s*→ ***Can***$_a$ *s*.

The rules listed above build the basic reasoning structure for the proposed formalism. By pursuing further about usage of quality attributes, other service composition/decomposition rules, formal models with richer expressiveness can be built, and analyzed. For instance, by explicitly representing quality requirements, we will be able to reason about how quality requirements can be used in service selection. By considering scenarios that actor tells false capabilities, knowledge, and beliefs out of malicious intent, we will be able to model trust issues in the service world.

3.2 主体服务场景模型举例

A world of two persons: A Simple Direct Service model

In a world of two persons, we assume that there is no third party and zero advance knowledge is available to either side of services. Conducting analysis to such models is to find another actor through whom the required services of an actor can be accomplished through *delegation*. The basic assumption is that a capable and trusted actor can be depended on for the fulfillment of a service request an actor has. The model shows the reasoning procedures of the two actors regarding a service situation SC_{20}:

In a physical world, knowledge about the participants of a service relationship can be obtained easily, for instance, Friday sees Robinson growing crops; so he believes that Robinson has the capability of providing food to him. Such scenario works fine in a closed world where people can meet face-to-face easily. However, when we come to an open world where direct observation and past experience are not available, how do we build a relationship between the service requestor and the service provider? What new problem do we need to deal with? Assume that now Robinson and Friday come

to the digital civilization, and get online, and assume their needs and capabilities remain the same.

A World with Deception: A Service Model on Trust

The publication rules set given in Rule 2.4 is based on an assumption that the actors in the system are telling the truth, but this may not be the case in the real world. Assume that there is an actor who lies about his capability to obtain another actor's service. We may extend the framework with action rules such as the following:

Publish false capability:

Actor *(a)* ∧ ***Actor*** *(b)* ∧ ***Service*** *(s)* ∧ *no* ***Can*** $_a$ *s* ∧ ***Know*** $_a$ *(* ***Requires*** $_b$ *s)* ⇒ ***tell*** *(a,* ***Can*** $_a$ *s, b).*

The service situation can evolve into the one represented by the following graphical model:

Establish black list:

Actor *(a)* ∧ ***Actor*** *(b)* ∧ ***Service*** *(s)* ∧ ***delegate*** *(a, s, b)* ∧ *no* ***perform*** $_b$ *s* ⇒ ***Know*** *(a, not* ***Can*** $_b$ *s).*

From this model we can see that the proposed formalism can be used to describe different domain assumptions, operational rules in a service environment. By analyzing the differences between systems showing desired properties, and those allowing undesirable behaviors, a designer will be able to build mechanisms reflecting the right control schema.

A World with Circle of Trust: Service Selection based-on Community Feedback

As mentioned in the previous section, in an open environment, direct knowledge about others actor is very hard to obtain. And sometimes, beliefs about another actor's trustworthiness on providing a service are not two value black-or-white assertions, but vectors using discrete values to represent varying levels of confidence. For instance, we may adopt a trust scoring schema to quantify the confidence level of beliefs circulated within the service network.

i. At the beginning, the trust level of all actors and beliefs is 0.

ii. Whenever an actor successfully delivers a service, its trust level to the service user will be increased by 1.

iii. When an actor fails to deliver a delegated service, its trust level will be decreased by 5 or to –1 whichever is higher.

iv. Whenever an actor recommends a provider who delivers a service successfully, its trust level to the service requestor will be increased by 1.

v. Whenever an actor recommends a provider who fails to deliver a service, its trust level to the service requestor will be decreased by 1.

vi. The confidence level of a recommendation is based on the recommender's confidence to the content, and the recommender 's confidence level to the receiver of the recommendation.

Naturally, we may consider defining a function of each of the beliefs in *B* of a service situation *SC*, whose domain is A ∪ B, with range being Integer:

Rule 2.10: **Trust Function Management Rules**

1. Set initial Trust value between actors (in response to rule (i) above):
   Actor *(a)* ∧ ***Actor*** *(b)* ∧ *no* ***Belief*** $_a$ *b* ⇒ *f(a, b) = 0*
2. Compute Trust value of a received recommendation (in response to rule (vi) above):
   ∃ *x*∈ *R*∪ *C*∪ *K*∪ *B*, ***Actor*** *(a)* ∧ ***Actor*** *(b)* ∧ ***tell*** *(a, x, b)* ⇒ *f(b, x) = f(b, a) × f(a, x).*
   ∃ *x*∈ *R*∪ *C*∪ *K*∪ *B*, ***Actor*** *(a)* ∧ ***Actor*** *(b)* ∧ ***tell*** *(a, x, b)* ∧ ***Belief*** $_b$ *x* ⇒ *f'(b, x) = f(b, x) × f(b, a) × f(a, x).*
3. Compute Trust after a service (in response to rule (ii, iii, iv, v) above):
   Actor *(a)* ∧ ***Actor*** *(b)* ∧ ***Service*** *(s)* ∧ ***delegate*** *(a, s, b)* ∧ ***perform*** $_b$ *s for a* ⇒ *f' (a,* ***Can*** $_b$ *s) = f (a,* ***Can*** $_b$ *s) +1 .*
   ∃ *x*∈ *R*∪ *C*∪ *K*∪ *B*, ***Actor*** *(a)* ∧ ***Actor*** *(b)* ∧ ***tell*** *(a, x, b)* ∧ *no* ***Perform*** $_b$ *s for a* ⇒ *f' (a,* ***Can*** $_b$ *s) = f (a,* ***Can*** $_b$ *s) –5, if f (a,* ***Can*** $_b$ *s) ≥ 4; f' (a,* ***Can*** $_b$ *s) = -1 , otherwise.*
   Actor *(a)* ∧ ***Actor*** *(b)* ∧ ***Actor*** *(x)* ∧ ***Service*** *(s)* ∧ ***delegate*** *(a, s, b)* ∧ ***perform*** $_b$ *s for a* ∧ ***tell*** *(x,* ***Belief*** $_x$ ***Can*** $_b$ *s, a)* ⇒ *f' (a,* ***x****) = f (a,* ***x****) +1 .*
   Actor *(a)* ∧ ***Actor*** *(b)* ∧ ***Actor*** *(x)* ∧ ***Service*** *(s)* ∧ ***delegate*** *(a, s, b)* ∧ *no* ***perform*** $_b$ *s* ∧ ***tell*** *(x,* ***Belief*** $_x$ ***Can*** $_b$ *s, a)* ⇒ *f' (a,* ***x****) = f (a,* ***x****) - 1 .*
4. Select a service according to trust level:
   Actor *(a)* ∧ ***Actor*** *(b)* ∧ ***Actor*** *(x)* ∧ ***Service*** *(s)* ∧ ***Requires*** $_a$ *s* ∧ ***Belief*** $_a$ ***Can*** $_b$ *s* ∧ ***Belief*** $_a$ ***Can*** $_x$ *s* ∧ ***tell*** *(b, s, a)* ∧ *f (a,* ***Can*** $_b$ *s) ≥ f (a,* ***Can*** $_x$ *s) ≥ 0*

⇒ ***delegate*** *(a, s, b).*

参考文献

1. E. Michael Maximilien, Munindar P. Singh. A Framework and Ontology for Dynamic Web Services Selection. IEEE Internet Computing, September/October 2004, pp.84-93.
2. L. Liu, et al. Strategic Capability modeling of Services. Proceedings of the 2nd Service-Oriented Computing and Consequences to Requirements Workshop, at the Requirements Engineering Conference, 2006. To appear.

设计实施计划

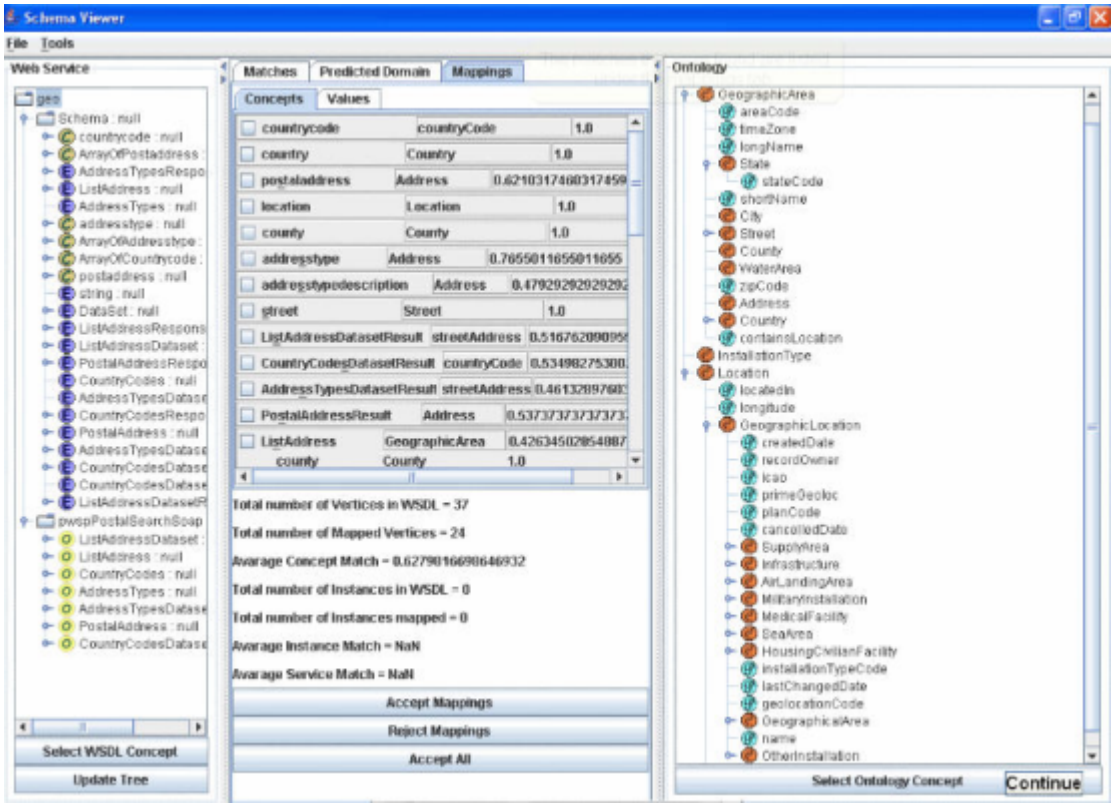
1. 选用的开源软件

METEOR-S

METEOR-S 是 Georgia 大学和 IBM Watson 研究院正在合作推进的研究项目。它是语义 Web 服务领域的一个轻量级解决方案(和 WSMO 和 OWL-S 比较而言)。目前我们关注它以下两个子项目:

- ✓ METEOR-S Semantic Web Service Annotation Framework (MWSAF) 负责把 WSDL 扩展为 WSDL-S，通过添加注释的手段给服务描述添加语义。另外，服务的选择和匹配算法也在这里实现，所以这个框架是我们关注的重点。
- ✓ METEOR-S Web Service Composition Framework (MWSCF)，这个框架负责如何把服务组合成流程。由于时间关系，我们还没有对这部分内容做仔细研究，如果能进复赛的话，我们会给出一个详细的服务组合方案。

下面是 MWSAF 的一个截图:



其中左边是 Web 服务的 WSDL 接口，右边是对应的本体，中间是应用匹配算法后计算出来的相似度。我们服务集市的实现将很大程度上依赖于这个框架。

METEOR-S 是一个和 IBM 合作的在研项目，虽然目前还没有明确的 License，但是显然不用担心授权的问题。

Jena

Jena 是 HP 公司推出的用来构建语义网应用的 java 框架，它提供了对 RDF，RDFS 和 OWL 的支持，并且支持若干个基于规则的推理引擎。Jena 框架包含以下组件:

- 操纵 RDF 的 java API
- 将 RDF 模型序列化为 RDF/XML，N3，N-Triples
- OWL API
- 模型既可以暂存在内存，也可以以文本或数据库的方式持久化
- RDF 查询语言 RDQL。我们主要用它从知识库获取信息

Jena 是我们设计语义化的服务集市平台的基础，它被用来创建，倒入，推理，持久化本体模型，并提供查询手段。

Jena 使用的是 BSD 风格的 License，符合大赛的要求。

Word-Net

Word-Net 是普林斯顿大学基于认知科学开发的词汇推理系统，所有的近义词都被字面的和语义的联系维持在一起。 Word-Net 对于我们的服务匹配算法有重大意义，是进行相似度分析的基础。

Word-Net 使用的 License 见附件，不具有传染性，符合大赛要求。

Protégé

Protégé是斯坦福大学开发的本体集成开发环境，最新的 3.2 版已经开始支持 OWL 表示的本体。配合众多威力强大的插件，Protégé可以大大减轻创建本体的难度，比如 OWLViz 可以支持可视化编辑 OWL 本体、Protege2Jena 可以把 Protégé 创建好的本体倒入到 Jena 持久层、Query Export Tab 可以直接用查询语言查询创建好的本体知识库等等。

Protégé使用 Mozilla Public License，符合大赛要求。

2. 选用的开源软件 IBM 产品

2.1 WebSphere 产品家族

WebSphere Application Server

作为 WebSphere 软件平台的基础，WebSphere® Application Server V6.0 是业界最主要的基于 Java™ 的应用程序平台，它集成了用于动态电子商务领域的企业数据和事务。每个可用的配置都交付了具有应用程序服务功能的丰富应用程序部署环境，这些服务提供了用于事务管理的增强功能，以及人们期来自 WebSphere

产品家族的安全性、性能、可用性、连通性和可伸缩性。这些可选产品提供:

- 完全兼容 J2EE V1.4，以及基于规范并超越规范的 Web 服务支持。
- 新的打包产品完全兼容 J2EE 1.4（从基本的 Express 版本到最健壮的 Network Deployment 版本），并提供了多个部署选项（从单台服务器到集群化、高可用性、高容量的配置）。
- 快速开发和部署特性，缩短了开发周期时间，最大限度地提高了使用现有技能和资源的能力。
- 紧密集成 IBM Rational 工具，是一个构建在 Eclipse 上的高效开发环境，也是一个开放的系统开发环境。
- 业界最广泛的跨平台支持。

WebSphere Application Server 的优势在于:

- **面向服务架构的构建块**: 现在 WebSphere Application Server V6.0 能够在所有配置上交付面向服务架构。这些产品允许企业使用基于标准的消息传递方式和最新的 Web 服务标准，通过重用现有 IT 资产来提高投资回报，并降低了总体拥有成本。从而降低开发成本、加速价值转化和增加业务灵活性。
- **安全的、最佳的资源利用**: WebSphere Application Server V6.0 为企业提供了安全而动态的平台。使用 WebSphere Application 的伸缩能力和安全特性，企业能够用较少的资源完成更多的工作。
- **随需应变的基础设施**: WebSphere Application Server V6.0 提供了当今企业最需要的健壮而灵活的环境。通过保证任务关键型应用程序接近连续正常运行，企业实际上可以避免失去业务机会。
- **快速开发和部署**: WebSphere Application Server V6.0 用增强的开发和部署能力来支持快速的价值转化。企业能够加速价值转化，并利用大多数现有技术技能和特性来支持易用而又高度开箱即用的性能。

WebSphere Application Server V6.0 可以用来作为本次大赛的应用开发平台。

WebSphere Process Server V6

WebSphere Process Server V6 是下一代业务流程服务器，它根据 SOA 和开放标准支持各种形式的集成，并使跨越人、工作流、应用程序、系统、平台和体系结构的业务流程自动化。IBM® WebSphere® Process Server V6 是一个基于 WebSphere Application Server V6 的全面的面向服务的体系结构的集成平台。可以使用 WebSphere Process Server 在面向服务的体系结构中开发和执行基于标准的、基于组件的业务集成应用程序。因为它是基于 WebSphere Application Server V6 提供的 J2EE 1.4 基础设施和平台服务的，WebSphere Process Server

包括诸如业务流程自动化之类的功能。您可以使用 WebSphere Integration Developer 开发运行于 WebSphere Process Server V6 的应用程序。

WebSphere Process Server 提供了四个服务组件：业务流程、人工任务、业务状态机和业务规则。可以使用 WebSphere Process Server 做以下事情：

- **事务、安全性、集群和工作负载管理**：WebSphere Process Server 解决方案使用 WebSphere Application Server 功能，因而向您提供了一个可伸缩的、可靠的业务集成环境，可以用于事务、安全性、集群和工作负载管理。
- **完整的 ACID 事务支持**：WebSphere Process Server 为业务流程提供了完整的 ACID 事务支持，既包括短时间运行的流程（端对端的一个事务）也包括长时间运行的流程（多个流程）。可以在工具中修改事务边界来将业务流程中的多个步骤集中到一个事务。此外，它还支持 WS-BPEL 规范中定义的业务流程的灵活补偿。
- **恢复管理器与恢复控制台**：WebSphere Process Server 包括恢复管理器和恢复控制台。如果在执行业务集成应用程序期间出现故障，则服务器将检测该故障，并允许您（管理员）在恢复控制台管理出错的应用程序。
- **封装业务功能**：WebSphere Process Server 中的唯一体系结构允许将业务功能封装到各个模块，然后单独进行更新。例如，您可以使用包含用于实际审批的人工任务的审批模块，随后使用包含业务规则的另一个审批模块替换它。这一更改对于该模块的使用者是完全透明的。此外，封装的概念确保了数据和接口定义在使用它们的位置封装。例如，可以隐藏如何在模块内的后端系统中表示使用者的细节，而模块本身将具有一般业务对象的通用接口作为数据公开。这一规范的数据表示还启用了任何给定集成应用程序中的高度重用。

WebSphere Business Integration Modeler

该工具家族扩展了建模的功能，并向建模世界交付了新的OPEN Systems可用性，在 ECLIPSE 上工作，并具有生成业务流程执行语言（BPEL）的能力。这个新的建模组合工具为刚开始建模方法的公司提供了一组新的功能。

分析师可以用来使用 WebSphere Business Integration Modeler 定义按需业务流程，其中由业务分析师进行的自顶向下的缄默是按需业务流程生命周期方法论的一个关键元素，BP 模型定义了技术框架来讲业务规范和 IT 开发联系起来，一个共享的模型在业务流程整个生命周期被预留，来帮助保持业务和 IT 观点同步。

WebSphere Developer Integration Edition

使用下一代集成开发环境来扩展和整合现有的 IT 资产，其中该环境经过了优

化，用来构建部署到 IBM WebSphere Business Integration Server Foundation 的组合应用程序。

- 通过从 Web 服务、Java 资产、后端系统和打包的应用程序、人员和过程中轻松创建可重用服务，从而最大程度地提高 IT 投资回报。
- 通过利用面向服务的架构来构建能够快速适应变更的模块化应用程序，从而提高 IT 的响应能力。
- 使用丰富的应用程序和技术适配器的广泛组合来扩展现有系统的范围。
- 通过使用拖放开发工具来快速构造新的基于过程应用程序，以便可视协调软件资产之间的交互，从而最大程度地提高开发人员的效率。
- 通过使用业务规则，将自适应业务逻辑嵌入应用程序和业务流程中，从而对变更作出预测。

通过构建行业领先、行业测试、行业支持的 WebSphere 平台来最大程度地降低开发、部署和管理成本，保护您的基础设施投资，并通过使用行业支持开放标准来开发应用程序，以便最大程度地降低培训成本。

2.2 Rational 产品家族

Rational Application Developer

使用这个完整的 IDE 可以快速设计、开发、分析、测试、配置和部署 Web、Web 服务、Java、J2EE 和门户应用程序。IBM Rational Application Developer for WebSphere Software 针对 IBM WebSphere 软件进行了优化，支持多供应商的运行时环境，得到了 Eclipse 开放源代码平台的强力支持，开发人员可以调整和扩展开发环境以适应自己的需要，提高生产率。如果与 IBM Software Development Platform 一起使用，开发人员可以直接从 Rational Application Developer for WebSphere Software 中访问各种需求和变更管理功能。

- 使用 RAD 工具和向导加快门户、SOA 和 J2EE 的开发。
- 采用拖放式的 UI 组件以及点击鼠标即可建立数据库连接，开发人员可以利用现有的技能，缩短了 Java 学习曲线。
- 使用针对编码标准视图、组件和 Web 服务单元测试、多层结构运行时分析的自动化工具改进代码质量。
- 把业务应用程序和可互操作的 Web 服务、面向服务的架构整合在一起。
- 通过 UML Visual Editor for Java and EJB 可视化和图形化地编辑代码。
- 使用内置的 IBM Rational ClearCase LT 版本控制在团队之间协作和共享资产。
- 使用基于 Eclpise 的插件调整和扩展开发环境以满足自己的需要。
- 使用拖放式 UI 组件和 Crystal Reports 快速构建和部署交互式报告。

Rational Software Architect

在一个开发团队中，软件架构师和高级开发人员要负责确定和维护应用程序架构的各个方面。他们需要功能强大、易于配置的工具来管理当今应用程序的复杂性。IBM Rational Software Architect 是一种集成的设计和开发工具，通过使用基于 UML 的模型驱动的开发，来创建结构更为合理的应用程序和服务。
借助于 Rational Software Architect，您可以将软件设计和开发的各个方面统一起来:

- 开发应用程序时比以前更加卓有成效。
- 利用建模语言技术中的最新成果。
- 检查和控制 Java 应用程序的结构。
- 利用开放的和可扩展的建模平台。
- 简化设计和开发工具解决方案。
- 与生命周期中的其他方面进行整合

Rational Web Developer for WebSphere Software

利用易学易用的 IDE 来构建、测试和部署 Web、Web 服务和 Java 应用程序。IBM Rational Web Developer for WebSphere Software 是为 IBM WebSphere 软件优化过的，并且能够支持其他供应商的运行时环境。它以 Elicpse 开源平台做后盾，因此您可以调整和扩展您的开发环境，以满足自己的需要并提高生产力。当与 IBM Software Development Platform 一起使用时，您可直接从 Web Developer 访问大量的需求和变更管理功能。

- 利用 RAD 工具和向导加速 Web、Java 和 SOA 开发。
- 通过拖放 UI 组件和点击数据库连接性，能够利用现有技能并缩短 Java 学习时间。
- 将您的业务应用与可互操作的 Web 服务和面向服务的架构进行整合。
- 通过集成单元测试环境和可视化调试器使应用程序测试流水线化。
- 利用基于 Eclipse 的插件调整和扩展您的开发环境，以满足您的需要。
- 通过与 IBM Software Development Platform 整合来跟踪活动，保证质量和管理版本。

3. 使用的 ERP 产品

用友 ERP-NC3.1

作为国内当前唯一能与国外厂商相抗衡的高端 ERP 产品，用友 ERP/NC 走过了一条有困难但又充满期望、另人激动的发展之路。国际管理模式、行业解决方案、先进技术架构是 NC 率先带给广大用户的真切感受，NC 也因此先后在 500

家企业集团得到成功应用。NC 在继承当前 ERP 的基础上，不断吸纳最新的、符合中国国情的先进管理思想或管理模式，增强集团化及全球化发展的适应性，满足电子商务环境下企业间协同和商务功能不断创新的需要。其先进性还体现在管理流程的可配置、基于知识的管理智能、实现企业的实时化集成，并通过满足企业组织或业务处理的动态调整来增强快速应变能力，借助快速实施、知识复制来实现管理价值的最大化。

NC3.1 的优点总结起来有以下几点：

- 持续创新与领先。
- 面向集团的管理与应用。
- 蕴含先进的管理理念。
- 具备先进的应用架构。
- 具备先进的技术架构。
- 拥有成熟的商业模式。

4. 使用的 CRM 产品

TurboCRM

目前国内 CRM 在理论和实践上日臻成熟，各厂商也纷纷推出自己的 CRM 解决方案，TurboCRM 就是其中一款优秀的 CRM 系统软件，它创造性的将 CRM 管理思想融入了软件设计之中，正是先进的管理理念和手段与软件技术完美融合，才使得 TurboCRM 成为出色的客户关系管理工具。TurboCRM 以其广泛的适用性、完善的功能、出色的个性化设计等诸多特性成为 CRM 软件中的一支劲旅。

TurboCRM 不仅仅是业务操作的工具，它从业务自动化，协同工作，客户关系提升，“知己知彼”和管理提升五个层面辅助企业全面改善客户关系：

- TurboCRM 实现了客户关系管理业务自动化。
- TurboCRM 实现了企业的协同工作。
- TurboCRM 帮助企业提升客户关系。
- TurboCRM 可以帮助企业“知己知彼”。
- TurboCRM 可以帮助企业提升整体管理水平。

SOA 大赛项目管理文档

1 简介

1.1 目的

此文档用于对清华大学软件学院 Zoo 团队参加 2006 “IBM 杯” 中国高校 SOA 创新应用大赛初赛期间的所有活动和注意事项进行统筹、规划和管理。

1.2 范围

此文档适用于 2006 年 4 月 11 号至 2006 年 6 月 24 号即整个 SOA 大赛初赛期间 Zoo 团队的学习研究以及初赛作品的分析、设计直至提交等一系列活动。

1.3 参考资料

参考资料主要包括本届 SOA 大赛 IBM 官方网站上推荐的学习材料，IBM DevelopWorks 中国网站上的相关技术文章，水木社区 “IBM 技术研究与开发” 版面上的相关文章以及 Zoo 团队自己找到的学习材料。

1.4 概述

本文档的第一部分是项目章程，主要包括项目综述，项目实施，风险管理，质量控制，项目进度，人力及时间估算，辅助设施及资源等几个部分；第二部分是项目范围说明，包括项目目标，可交付成果，里程碑，技术要求，限制和排除，与客户共同检查等几个方面；第三部分是工作分解结构，对整个初赛阶段所要做的工作进行详细分解。

2 项目综述

2.1 背景

凤凰医疗设备有限公司是一家专门制造和营销专业医疗器械和实验仪器仪表等仪器的民营企业，其购销客户和网络遍布全国各地。凤凰成立于 2000 年，现有员工 750 名。公司领导一直非常重视企业信息化建设，投入大量的资金支持，并制定"把握趋势、兼顾现实、统一规划、逐步实施"的发展策略。

2004 年凤凰公司引进并在公司内部成功实施了某 ERP 系统（部署在凤凰企业内部的 Web 应用），主要用于凤凰公司的财务管理，其中包括产品库存及订单管理等。ERP 的实施大幅度地提高了公司的管理效率。随着公司业务规模的扩大和产品质量的提升，凤凰公司的客户数量越来越大。凤凰公司有一批精干的销售队伍，他们经常出差和客户打交道。虽然销售人员都配备了笔记本电脑，使

他们能够方便地和公司通过 email 发送和接受文档，但是竞争的压力使得凤凰公司不得不考虑使用客户关系管理系统（CRM）来进一步提高销售人员的工作效率。于是，2005 年 8 月份凤凰公司引进并在企业内部成功实施了某客户关系管理系统。凤凰的销售人员在任何时间和地点只需要连接企业内部网，并通过普通的 Web 浏览器就可以使用和管理客户及销售信息，包括客户信息，商机，业务机会，以及客户及销售信息分析图表等。

现在凤凰公司的财务和销售人员分别在 ERP 和 CRM 系统上工作，工作效率有很大提高。但是公司目前也面临挑战。一方面，ERP 和 CRM 中分别维护产品和客户信息，而公司规定 ERP 必须作为这些信息的主数据源，ERP 中的这些信息需要随时同步到 CRM 中去；另一方面，CRM 中维护的业务机会和 ERP 中维护的销售订单有着非常紧密的关系，凤凰公司希望能够把业务机会和销售订单有效地整合起来，而进一步提高业务运作的效率。

2.2 业务需求

基本的业务需求即为凤凰公司的 CRM/ERP 业务整合需求，其中 CRM 系统的基本功能包括: 创建客户和联系人信息，创建业务机会，更新销售的产品条目，同步产品信息，更新业务机会状态，发送销售订单处理请求等；而 ERP 系统的基本功能包括: 财务人员工作任务通知，财务人员处理销售订单，建立或修改客户账户，客户账户 Email 有效性校验，创建/修改销售订单，更改 CRM 信息，通知销售人员销售订单的处理结果等。我们需要借助于 SOA 的思想、体系架构及相关技术实现 CRM 系统与 ERP 系统的业务整合。

除了基本的业务需求之外，凤凰公司还有一些扩展的业务需求，包括: 可视化的信息聚合要求，信息智能服务等。上述业务需求只是基本的需求框架，这些业务需求在实现中均是可选的，可以选择不实现某些需求，也可以修改或添加更富有创意的需求并实现之。

2.3 项目意义及影响

该项目的实施能够整合凤凰公司内部的 ERP 系统和 CRM 系统的业务需求，有效地维护产品和客户信息，把业务机会和销售订单有效地整合起来，而进一步提高业务运作的效率，提高企业的管理效率，有助于提升凤凰公司的企业信息化程度，推动该公司“把握趋势、兼顾现实、统一规划、逐步实施”的发展战略的尽快实施和实现。

2.4 项目主要干系人

项目的主要实施人员为清华大学软件学院 Zoo 团队的五位成员，包括领队刘璘老师和团队成员乔伟、向坚、江铁扣和熊哲。

2.5 项目的批准和授权

该项目主要由 Zoo 团队领队刘璘老师于 2006 年 4 月 10 号批准，授权乔伟为项目经理，并于 4 月 11 号起正式实施该项目。

3 项目实施

3.1 项目的交付成果及质量目标

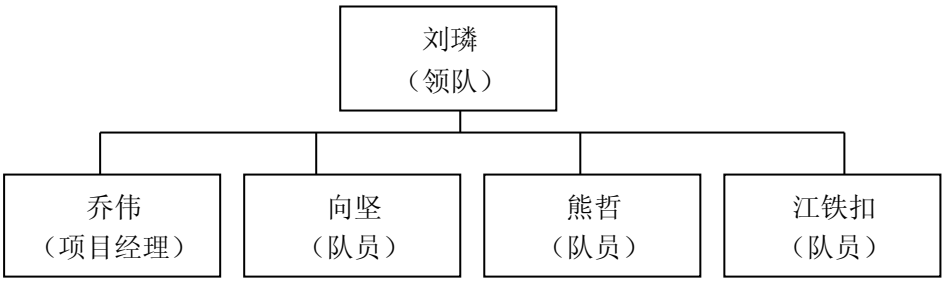
该项目的必须交付件包括：交付件清单（要求说明每个交付件的用途和使用方法等），设计文档（系统架构，组件设计），系统开发计划安排，解释 SOA 思想和方法在本系统开发中的应用。

该项目的可选交付件包括：业务模式分析和设计（包括商业价值分析），服务模型分析和设计，解释 SOA 思想和方法在本系统开发中的应用。

该项目的首要质量目标即满足凤凰公司的基本和扩展业务需求，同时要有自己的创新点，不仅要通过初赛，还要争取进入决赛，拿到冠军！

3.2 组织机构及责任

3.2.1 组织结构图



Zoo 团队组织结构图

3.2.2 Zoo 团队成员职责划分

清华大学软件学院 Zoo 团队		
成员姓名	昵称	职责
刘璘（领队）		负责整个团队的技术指导和协调工作

乔伟（项目经理）	熊（Bear）	负责整个项目运行期间的管理和协调工作，是整个项目的总负责人和技术总监
向坚（队员）	大象（Elephant）	负责整个系统的分析以及具体模块和细节的实现
熊哲（队员）	猴子（Monkey）	负责设计系统的体系架构，同时担任项目运作期间 Zoo 团队同外界交流的联系人
江铁扣（队员）	青蛙（Frog）	负责项目早期的需求分析与建模以及整个项目实施过程中的技术支持和文档编制

3.3 项目生命周期的主要阶段

在整个项目的生命周期即 2006 年 4 月 11 号至 6 月 30 号之间，整个项目将主要包括以下几个阶段：学习研究大赛相关的学习资料阶段；业务模型分析设计阶段；用例模型分析设计阶段；成熟度 SIMM 分析阶段；业务流程分析设计阶段；服务分析建模阶段；组件分析设计阶段；服务实现阶段；制定系统实施计划阶段和提交初赛作品阶段。

3.4 风险管理

风险名称	发生阶段	发生概率	影响	应变计划
成本超支（主要指时间）	整个项目生命周期	高	导致正常的课业学习受到影响	对每项任务做出合理的计划并确保按时实施
团队成员离开	整个项目生命周期	低	导致整个项目的进度及相关任务受阻	尽块邀请其它合适的同学加入团队中
遇到技术难题难以攻克	整个项目生命周期	中	导致项目的某一部分进度延缓甚至不能完成	集体讨论学习，请教老师或其它同学
各功能模块不能很好地整合	系统即成阶段	高	导致项目的功能不达标甚至使得整个项目	在项目初始阶段对系统的架构和各部分功

			失败	能进行深入的分析和研究，制定整合策略

注：在上表中，将风险发生的概率分为高、中、低三种情况，影响表示风险一旦发生会对项目本身或项目成员产生的影响，应变计划表示风险发生后应该采取的补救措施。

3.5 质量控制

主要采取以下几种措施对项目的质量加以控制：

（1） 在项目的初始阶段对系统的功能、架构及其它细节进行详细的分析和讨论，制定详细的项目实施计划；

（2） 对系统的每一个功能模块都要明确分配到个人，由该成员全权负责该模块的分析、设计和实施，最后由全体成员加以讨论和评估；

（3） 遇到技术难题时由所有成员加以讨论和分析，选择一种最为合理的应对策略，尽快攻克该难题；

（4） 对所有成员的工作和职责进行明确的划分，各成员的工作完成情况要经常同全体成员进行汇报、交流和讨论；

（5） 每位成员都要对各自所作的工作进行相应的记录并保存在特定文档中，以便其它成员了解整个项目的进度和完成情况，也有利于该成员对此前的工作进行修改和完善；

（6） 全体成员的工作都要符合统一的形式要求，在此基础上发挥各自的技术特点和优势；

（7） 定期对已完成的系统模块进行检查审核，做出相应的返工、修改、完善或集成等决定；

（8） 不定期地进行单元测试或集成测试，特别是在完成整个系统的集成之后，要按照客户的要求对系统功能进行测试，以确定系统是否实现了相应功能并满足客户的需求。

3.6 人力及时间估计

在项目的整个生命周期中，所有的人力资源即为 Zoo 团队的五位成员，其中领队刘璘老师主要负责项目的技术指导和协调工作，而另外四位团队成员则负责项目整个生命周期中各个阶段的分析、计划、实施和检查检验等工作，预计每人每天平均将花费 2--3 小时的时间。

3.7 辅助设施及资源

完成该项目的辅助设施主要为电脑，所需的资源包括 IBM 公司提供的相关软件及其推荐的相关学习资料，还包括 IBM DevelopWorks、水木社区“IBM 技术研究与开发”版面上的技术文章，以及 Zoo 团队自己搜集的相关学习资料。

4 项目范围说明

4.1 项目目标

Zoo 小组（共五人）在 2006 年 4 月 11 日到 2006 年 6 月 30 日之间利用课余时间研究 SOA、ERP 和 CRM 等相关知识，在 6 月 30 日之前完成并提交 2006“IBM 杯”中国高校 SOA 创新应用大赛所要求的初赛交付作品。

4.2 可交付成果

（1）必须的交付件

交付件清单（要求说明每个交付件的用途和使用方法等）;

设计文档（系统架构，组件设计）;

系统开发计划安排;

解释 SOA 思想和方法在本系统开发中的应用。

（2）可选的交付件

业务模式分析和设计（包括商业价值分析）;

服务模型分析和设计;

解释 SOA 思想和方法在本系统开发中的应用。

4.3 里程碑

（1）学习研究大赛相关的学习资料，4 月 30 号;

（2）业务模型分析设计，5 月 10 号;

（3）用例模型分析设计，5 月 17 号;

（4）成熟度 SIMM 分析，5 月 20 号;

（5）业务流程分析设计，5 月 27 号;

（6）服务分析建模，6 月 3 号;

（7）组件分析设计，6 月 10 号

（8）服务实现，6 月 17 号;

（9）制定系统实施计划，6 月 24 号;

（10） 提交初赛作品，6 月 27 号。

4.4 技术要求

（1）基于 SOA 的思想理念，实现凤凰公司 ERP 系统和 CRM 系统的整合;

（2）要体现 SOA 的特点，包括：IT 的实现是否考虑了业务的需求；IT 的实现是否能够适应业务需求的变更；组件的可重用程度等;

（3）必须具备创新性，包括：作品的总体创意和原创性；创新性可以体现在业务模型的创新、IT 技术的创新等等各个方面，包括 WEB 2.0 技术的使用;

（4）平台的选择要合理，包括：是否合理地选择了 IBM 的产品；是否合理地选择了开源软件；

（5）所提交作品的完整性和规范性要好，包括：是否涵盖了提纲所要求的所有基本内容；方案描述、辅助图表是否准确清晰；技术用于和整体文字组织是否准确精炼等。

4.5 限制和排除

（1） 必须注重业务需求和 IT 的一致性；

（2） 业务需求在实现中是可选的，可以选择不实现某些需求，也可以修改或添加更富有创意的需求并实现之；

（3） 尽可能重用已有系统的功能；

（4） 可以选择使用 IBM 的产品族以支持整个 SOA 解决方案的生命周期，包括 Rational,Websphere,DB2 等系列产品；

（5） 可以在开发和运行中使用开源软件，但是必须遵循如下 License 的限制：

禁止使用任何版本的 GPL/LGPL License 的开源软件；

可以使用 Eclipse、Apache、CPL、BSD 等 license 的开源软件；

5 工作分解结构（WBS）

1.0 “IBM 杯”中国高校 SOA 创新应用大赛初赛

1.1 学习研究大赛相关的学习资料

1.1.1 学习 SOA 的基本概念

1.1.2 学习与 SOA 实施相关的概念、技术和软件

1.1.2.1 学习企业服务总线（Enterprise Service Bus）

1.1.2.2 学习业务流程执行语言（Business Process Execution Language）

1.1.2.3 学习 WebSphere Process Server 及 WebSphere Integration Developer

1.1.2.4 学习 Rational Software Architect 及 Rational Application Developer

1.1.3 学习 ERP 相关知识及用友软件

1.1.4 学习 CRM 相关知识及 TurboCRM

1.2 业务模型分析设计

1.2.1 业务组件建模（CBM）分析与设计

1.2.2 企业架构（EA, Enterprise Architecture）分析与设计

1.2.2.1 设计活动模型 Activity 图

1.2.2.2 设计信息模型 ER 图

1.2.3 电子商务模式（E-Business Pattern）分析与设计

1.3 用例模型分析设计

1.3.1 制定用例描述文档

1.3.2 设计用例模型并画出用例图

1.4 成熟度 SIMM 分析

1.4.1 进行 SOA 成熟度分析

1.4.2 书写成熟度分析报告

1.5 业务流程分析设计

1.5.1 凤凰公司业务流程建模

1.5.2 进行业务流程分解

1.6 服务分析建模

1.6.1 服务发现

1.6.1.1 流程分解和业务流程分析

1.6.1.2 业务目标定义和建模

1.6.1.3 自上而下分析现有系统

1.6.2 建立服务目录，包括所有可能的服务候选列表

1.6.3 建立服务规约，包括服务属性、使用、关联目标和业务逻辑等

1.7 组件分析设计

1.7.1 分析 ERP 系统结构图并划分功能模块

1.7.2 构建数据流程图和系统结构图

1.7.3 进行组建分解和组建划分

1.7.4 书写组建文档

1.8 服务实现

1.8.1 分析企业服务总线（ESB）及其参考架构

1.8.2 服务包装及新服务的编写

1.8.3 构建服务实现实例图

1.8.4 书写服务实现文档

1.9 制定系统实施计划

1.9.1 设计队伍的构成和分工

1.9.2 确定需要用到的软硬件环境、平台和工具

1.9.3 制定项目实施的任务分解和时间表

1.9.4 设计风险分析

1.9.5 设计验证和测试草案

1.9.6 确定作品演示的初步方案和所需要的环境设备支持

1.10 提交初赛作品

6 项目进度计划

项目的进度计划初步安排如下:

4 月 11 号至 4 月 30 号，团队所有成员共同学习所有与项目相关的资料和技术;

5 月 1 号至 5 月 10 号，由乔伟和向坚完成业务模型的分析与设计;

5 月 11 号至 5 月 17 号，由江铁扣和熊哲完成用例模型的分析与设计;

5 月 18 号至 5 月 20 号，由乔伟和向坚完成 SOA 的成熟度分析;

5 月 21 号至 5 月 27 号，由向坚和熊哲完成业务流程的分析与设计;

5 月 28 号至 6 月 3 号，由江铁扣和乔伟完成服务的分析与建模;

6 月 4 号至 6 月 10 号，由乔伟、向坚和熊哲完成组件的分析与设计;

6 月 11 号至 6 月 17 号，由乔伟、向坚和江铁扣制定服务的实现策略;

6 月 18 号至 6 月 24 号，团队全体成员共同商议制定系统的实施计划;

6 月 25 号至 6 月 27 号，团队所有成员共同检验检测初赛作品，由江铁扣提交给大赛组委会。

7 项目完成后的回顾总结

在项目的实施过程中，主要遇到以下几个方面的困难:

1. 五月份期间，团队的每一位成员都需要单独完成一篇论文，这对整个项目进度都产生了很大的影响，受到影响的工作包括业务模型的分析与设计、用例模型的分析与设计、SOA 的成熟度分析以及业务流程的分析与设计，造成这几项工作的实际完成日期比预计完成日期都有了不同程度的延迟;
2. 在项目实施的前期，团队成员之间容易出现交流和协作困难，除了公共的学习资料之外，每人单独找到的学习资料难以共享，另外每人的学习进展和所遇到的技术难题也难以及时地沟通和交流，在完成各项工作时也难以有效地协作，互相之间不是很清楚其他成员的工作进展;
3. 由于每人本学期都有几门课程要学习，而每门课程都有相当多的试验和大作业要完成，这就使得每位成员在某些时间段不能在 SOA 投入足够多的时间，这也会对项目进度产生负面的影响。

针对以上问题，经团队成员集体讨论，制定了相应的对策以保证项目的最终成功完成:

首先，为了解决团队成员写论文对项目造成的延误，我们决定将所延误的各个项目加以细化，使每个成员承担的工作可以单独完成，从而使被延误的这些工作可以并行进行，加快项目的实施进度，弥补延误的时间;

其次，为了解决团队成员之间交流不力的问题，我们决定采用 TortoiseSVN 版本控制软件，对所有的学习资料、每位成员的工作计划和任务分配、每位成员已完成的工作以及项目的总体进展进行控制，大大提高了每位成员的工作效率，也使得项目的进展大大加快;

再次，为了解决课业任务同项目实施在时间上的冲突，我们决定每天抽出固定的时间段来专门投入到项目工作的完成中去，确保项目进展不受其他事情的干扰，从而使得项目得以按计划顺利进行;

最后，团队所有成员经过两个半月的紧张奋战，成功解决了所遇到的时间、技术等各方面的难题，顺利实现了预期目标，完成了初赛所要求的所有交付件，衷心希望我们所有的努力和付出都能得到相应的回报，We are going for the champion!

后记：关于 SOA，关于我们

不知道这是不是关于结局的文字，或者，它还是又一次征程的檄文。

How many roads must a man walk down, before we call him a man … The answer my friend, is blowing in the wind.

一个男人，究竟要经历什么样的风雨，才会真正的成熟；

一群男人，究竟要经历什么样的旅程，才会真正的成长为一个团队。

答案，或许真的只是在风中飘，但是，对于我们来说，答案，在每一个人的心中。

我们与 SOA 结缘于 10 个月前，当时我们 4 个人被分到刘璘老师领导的清华大学知件研究组（Knowware Group）。就是在那个时候，SOA 第一次走进了我们的生活。随后我们的研究方向逐渐确立：bear 负责 Web Services 语义本体和信任机制的研究； elephant 负责用 i-star 方法对 SOA 的安全性进行研究； monkey 则是用 e-learning 作为案例研究 SOA 的领域相关模型；frog 负责用 MDA 的方法把我们的模型进行映射。我们是整个学院唯一每周开两次例会的团队，虽然当时不太理解并偶有抱怨，但是现在每个成员都非常感谢刘老师带我们进入 SOA 的大门并领略其中的无限风光。

好的开始，是成功的一半。作为 SOA 新鲜人的我们，这学期又有幸选修了 IBM 中国研究院和清华自动化系合作开设的信息服务，就这样获得了进入风光旖旎的广阔天地的机会。感谢 Business Optimization team 的曹荣增和田春华先生，他们对 Business Strategy & Design 的介绍使我们认识到 SOA 绝不仅仅是技术问题，还有 CBM 方法对于业务建模是多么重要； 感谢孟繁晶女士对企业架构的精彩讲解，使我们留连于 EA 的精彩世界，原来 SOA 之上还有 SOE，SOC 等层次，看待企业的角度居然可以这么宽广；感谢朱俊先生带来的关于业务流程建模的介绍，他用 IBM 的真实案例向我们讲解了怎样使用 WBI 进行 BPM 建模; bear 在这里要特别感谢潘岳博士，潘博士讲解的 Semantic Web 和 bear 的研究方向非常契合，在课下也对 bear 进行了 WSMO、OWL-S 等语义 web 服务内容的指导，坚定了 bear 把 SOA 语义化进行到底的决心； 最后我们要感谢陈颖先生，他对 ODOE(On Demand Operating Environment) 的介绍使我们对服务的运行时管理有了充分的了解。

SOA,就这样在我们的生活中立体起来，紧接着，就是 SOA 大赛。

从四月报名参赛，到五月初的资料收集与准备，到五月中的详细计划制定，再到五月底和六月的辛勤工作，一直到现在——提交的时刻，我们度过了丰富的

三个月；从最初的万事皆无头绪，从空白的 blog，从寝室里我们无精打采的脸，到 6 月 19 日第一百次提交和 1247 个文件，到四页的 blog，还有大家疲惫而开心的笑，这次比赛的经历，对于我们来说，更像是一次曲折的旅程。

我们四个人的性格，丰富而且天然的互补。Bear 沉稳大气，求知欲强并且有深厚的管理学背景；Elephant 是个天生的实干家，总是在无声无息之间高质量做好自己的所有工作；Frog 脚踏实地，有坚强的性格和持久的恒心；Monkey 积极乐观，思维活跃，善于与人沟通，富有创造力。

然而，team is not born to be, team is annealed to be.

SOA 大赛，于我们而言，是一个项目，我们是项目组里的成员。或许我们作为同学和朋友，在生活之中早就已经熟识，而作为一个 team，我们依然有长长的一段路要走。

抛开具体的比赛内容不谈，我们对于团队组织形式的思路明晰而坚定。我们始终要作为一个团结并且斗志昂扬的团队去工作，而 SOA 大赛，就是我们这个项目组的第一个大的项目。

事情却并非想象那样简单。我们每个人拥有各自独立的世界，我们的学习生活有着不同的时间表。课程，学校的活动，实验室的任务，自己的追求，所有的这一切交织在一起，使得我们很难抽出比较长的时间共同工作。在开始的阶段，我们每个人没有明确的任务，或者任务没有明确的 deadline，我们似乎一直在为 SOA 而奔忙，而我们又似乎总是在原地打转。

某一天，在 Google Talk 上，Elephant 问 Bear,我们的 SOA，究竟何去何从。

那一天晚上，我们在西门烤翅一起喝酒，昏暗的灯光下面，或许是同样昏暗的脸。

我是一只小小小小鸟，想要飞呀飞却飞也飞不高，或许那个时候的我们，就是这样的心情。

天将降大任于斯人，必先苦其心智。然而，也正是困难和不顺，才能增益我们所不能。

从项目管理的角度，我们仔细的思考团队缺少足够的行动力的原因：

我们缺少一个“项目经理”，每个人的职责划分的过分民主而没有效率；

我们缺少足够的共同工作的时间；

我们凝聚力高但是有些时候似乎丧失了批评性评价的能力；

总而言之，我们有了项目，但是我们得团队似乎却不是为这个项目而准备的。

知道了原因，我们就必须行动起来。于是事情出现了转机：Bear 成为了事实上的项目经理；在共同讨论的基础上，他辛苦的为大家分配任务和制定计划；大家把晚上的九点开始的时间段奉献出来用于讨论；大家每天都尽量的在实验室，随时交换意见和信息；。。。。。

SOA 大赛，或许还有 IBM，我们来了！

接下来就是大家脚踏实地的工作。有关初赛交付件的文档，一点一点的多了起来。第一次提交，第二次提交，...，第一百次提交，第一个文件，第二个文件，...，第一百个文件，...，第一千个文件，...，第一千两百个文件。不积跬步，无以致千里；不积小流，无以成江海。我们的 SOA 初赛之路，就这样一步步地走过。

在写这篇 SOA 后记之初，我们的胸中，似乎都藏了千言万语，每个人都想写下波澜壮阔的诗篇，关于这次 SOA 初赛的壮丽诗篇。按照我们的计划，这篇后记，本来应该像大海一样充满汹涌的激情。转眼之间，真到了动笔的时候，每个人都发现，居然激动不起来，情绪的大海，流诸笔端，居然掬成了平平淡淡的细水长流。或许繁华落尽，真的就是平淡了吧。幸运的是，我们比任何时候都知道团队的重要性，比任何时候都知道我们是一个团队。有一首歌叫做《一起吃苦的幸福》，这一次每个人真的都体会到了其中的滋味，或许，这也是我们在此次比赛中的收获之一。

现在写下这篇短文的时候，我们的心中，都有一个小小的希望，就是希望我们 Zoo 这个小小的团队可以再前进一步。我们还希望能够继续体会作为一个团结团队成员的幸福，一起为同一个目标打拼的幸福，一起吃苦的幸福。

最后写下我们都非常喜欢的歌词作为结尾 ——它凝聚了我们对过去的追思和对未来的憧憬。We believe we can fly!

I used to think that I could not go wrong
and life was nothing but that an awful song
but now I know the meaning of true love
I am leaning on the everlasting arms

if I can see it
then I can do it
I believe I can fly
I believe I can touch the sky
I think about it every night and day
spread my wings and fly away
I believe I can soar
I see me running through that open door

I believe I can fly

if I just spread my wings
I can fly
I can fly
I can fly
Oh if I just spread my wings
I can fly
I can fly

谨以此文纪念 SOA 初赛的日子

Zoo
2006 年 6 月 25 日