使用 Rational Unified Process 开发大规模系统

Maria Ericsson

Rational Software 白皮书

TP 156

Rational®
the software development company

目录

历史记录

这本白皮书描述“互连系统中的各个系统”，它由 Ivar Jacobson、Karin Palmkvist 和 Susanne Dyrhage 发表在 1995 年 5-6 月份的 ROAD 上 [1]。本白皮书受益于来自数个大规模系统开发项目的输入，并且也旨在与 Rational Unified Process V5.1 [2] 和统一建模语言 [3] 相一致。

互连系统组成的系统

当开发大规模系统时，复杂性会有相当程度的增加。它不仅要求您能够理解更复杂的一组工件，而且要引入开销，因为需要管理更大的一组资源。本白皮书描述了用于帮助控制增加的复杂性开销的体系结构模式。此体系结构模式与在 [4] 中讨论的其它位置一起被称为**互连系统组成的系统**。

当构建非常大型或复杂的系统（如命令和控制系统或高度集成的 IT 解决方案）时，此构造是很有用的。在大多数情况下，这类“超级系统”将被划分成几个单独的部分，将每一部分独立开发成一个单独的系统。超级系统由一组互连系统实现，彼此进行通信以履行超级系统的职责。这些系统中的一个表示总体功能，我们称之为**上级系统**。其它系统表示整个系统的一部分，我们称它们为**下级系统**。上级系统与实现它的下级系统显然是完全不同的。不同类型的系统之间的关系造成了这种不同：从上级系统的角度看，下级系统是子系统，请参阅图 1。

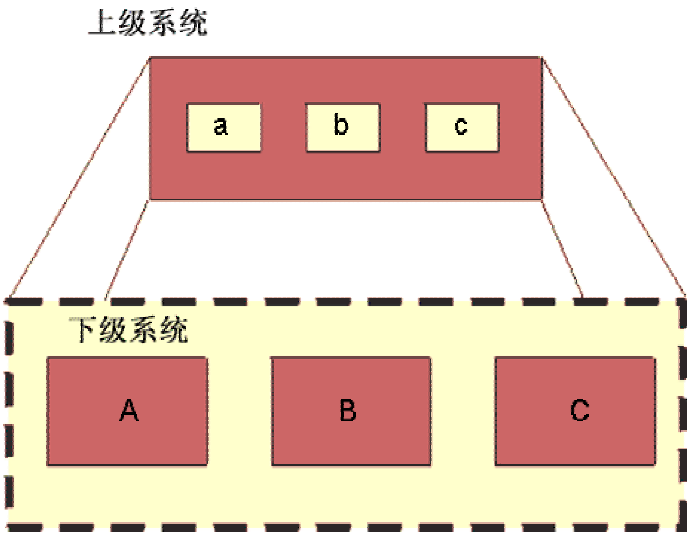

图 1. 上级系统的规范是通过互连系统组成的系统实施的，其中，系统 A、B 和 C 分别是上级系统的子系统 a、b 和 c 的实施。

将上级系统从下级系统中分离出来有几个好处：

- 在所有生命周期活动（包括销售和交付）期间，可单独管理下级系统。
- 它使得很容易使用下级系统来实现其它上级系统，方法是将下级系统插入互连系统组成的其它系统。
- 当开始建立系统时，您不总是知道它是否是一个互连系统组成的系统。您可以从一个“简单”的系统视图开始工作，并在生命周期中相当晚的时期确定是否需要应用互连系统组成的系统模式。
- 它允许您对下级系统做出内部更改，而不必开发上级系统的新版本。仅当发生重大功能性变更时，开发上级系统的新版本才是必需的。

每个下级系统都有一组相关工件，各工件之间具有明确的可跟踪性。同时也维护下级系统的工件集与对应的上级系统工件集之间的可跟踪性。可将每个下级系统作为一个单独的开发项目（具有自己的生命周期阶段：先启、精化、构建和产品化）来管理。

如果您正在建立的“超级系统”非常大，则可能需要进一步划分下级系统并因此将其视作一个互连系统组成的系统。

软件开发生命周期

在 Rational Unified Process 中，从两个角度简介和讨论开发生命周期：管理角度和开发角度，请参阅图 2。

从管理角度来看，经历四个生命周期阶段来开发系统或系统的新一代。从开发角度来看，以迭代方式开发系统的各个版本，这些版本以渐进方式越来越完整。迭代期间执行的活动已在 Rational Unified Process 中组成了一组核心工作流程。每个核心工作流程侧重于描述系统的某一方面，得出系统模型或一组文档。

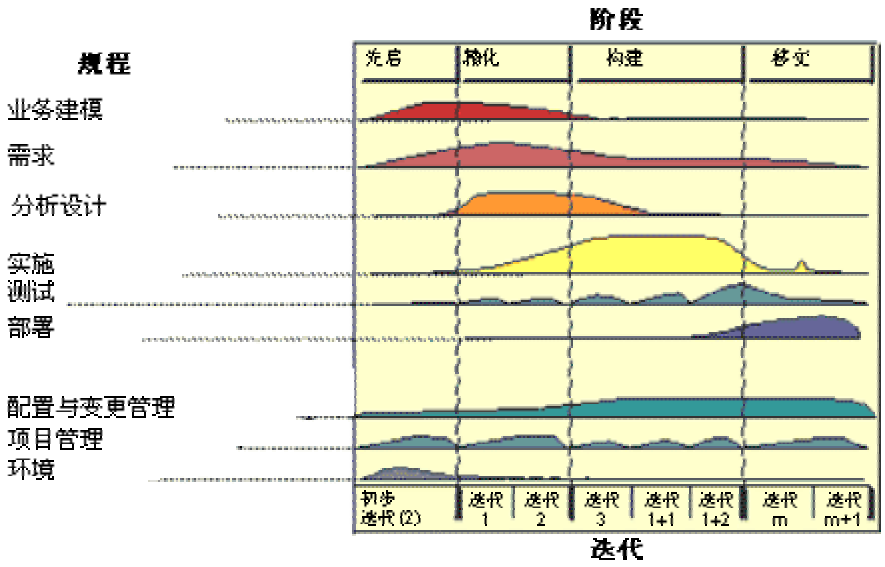

图 2. 迭代模型

将此模型应用于互连系统组成的系统，上级系统以及它的每个下级系统都会经历各自的生命周期，并通常被视为独立的项目。

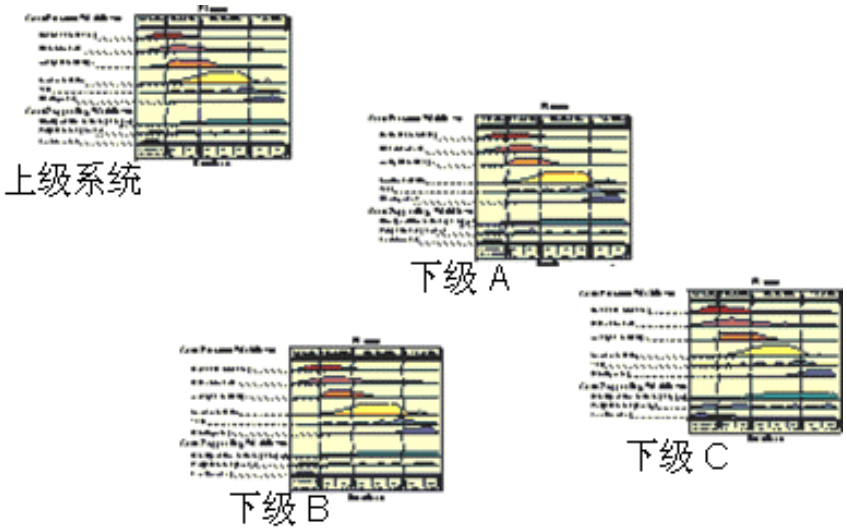

图 3. 每个系统（上级系统和下级系统）都经历各自的生命周期。

当然，生命周期具有依赖关系。正确管理那些依赖关系是开发互连系统组成的系统的挑战之一。有以下类型的依赖关系：

- 生命周期在时间上有依赖关系。上级系统的生命周期首先开始。一旦上级系统至少经过一个迭代并且下级系统的接口是相对稳定的，则可以开始下级系统的生命周期。事实上，在上级系统上经过至少一个迭代之前，您甚至可能不知道哪些是下级系统。
- 一旦下级系统的接口是稳定的，上级系统的生命周期就可进入维护阶段。这意味着不进行主动的开发，除非出现问题，要求变更下级系统的接口。
- 下级系统的接口是由开发上级系统的那些人员所有的。有关接口的更多信息，请参阅 [3] 和 [5]。
- 实现下级系统接口的类由开发下级系统的那些人员所有。

系统开发的工作流程和工件

一般的假设是上级系统及其下级系统可用同一组工件开发，并可通过通常用于非合成系统的相同工作流程来开发。在可以继续显示如何做到这一点之前，必须简介这些工件和工作流程。在 Rational Unified Process 中，我们简介五个核心处理工作流程，请参阅图 4。这些工作流程是：

- 业务工程 — 目的是评估要使用系统的组织，以更好地了解将由系统解决的需要和问题。结果是业务用例模型和业务对象模型。此工作流程可视为可选的。如果要使用系统的组织过于简单，则可能不会增添价值。
- 需求 — 目的是捕获和评估需求，注重可用性。这会产生用例模型，使用参与者表示与系统通信的外部单元，并使用用例表示事务序列，为参与者得出可评估的值结果。
- 分析与设计 — 目的是调查预期的实施环境以及将对系统构建产生的效果。这会产生对象模型（设计模型），包括显示对象如何通信以执行用例流的用例实现。这可能包括类和子系统的接口定义，根据提供的操作指定它们的职责。根据实施语言、分发等，此对象模型也适应于实施环境。考虑单独模型（又称为分析模型）的分析结果，有时是有用的。

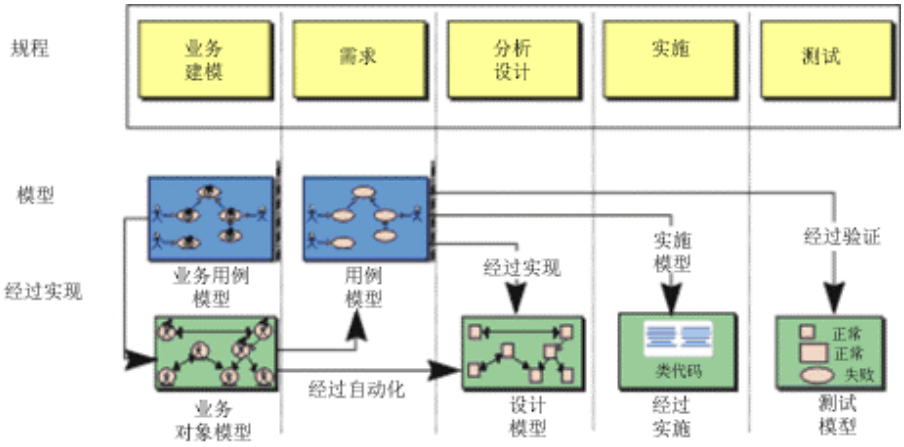

图 4. 每个核心处理工作流程与特定的一组模型关联。

- 实施 — 目的是在规定的实施环境中实施系统。这会产生源代码、可执行文件和文件。
- 测试 — 目的是确保该系统是预期的系统，并且在实施中没有错误。这会产生经认证的系统，该系统已准备就绪，可以交付。

互连系统组成的系统的开发

我们真正必须做的是定义如何将系统的职责分配给多个系统，每个系统负责这些职责的一个定义良好的子集。这表示主要目标是定义这些下级系统之间的接口。当完成此项工作时，剩下的工作可根据“分而治之”原理，对每个下级系统单独进行。因此，这就是总体上必须对系统执行的全部工作，此外一旦完成实施还要进行测试。

分解条件

如此一来，如何确定是否要将您的系统分解成互连系统组成的系统呢？有一些特征应该加以考虑：

- 对于具有相当规模和复杂性的系统，可以将问题分为更易一次理解一个的若干较小部分。
- 正在处理物理上独立的系统吗？通常使用旧系统或旧体系结构时会是这种情况。
- 分解有助于在系统各部分之间定义普通和有限的接口。
- 您可以决定使用某个主要的 COTS（商用成品）产品来实现系统的某一部分。分解将有助于阐明您打算如何使用 COTS 产品。
- 分解可使您获得分布式开发组织的最多好处，并且能够在多个地理上分散的团队之间明确划分工作。

要考虑到的风险是：

- 过度使用分解会掩盖所有问题细节的整体问题。
- 使用物理上独立的系统或物理上独立的团队时，您面临着破坏任何形式的重用的风险，并将以强硬的系统“卡壳”而告终。

组织

要缓解上面提到的风险，指定一组人员监督整体开发工作是很关键的。该小组通常被称为体系结构团队并应侧重于以下关键问题：

- 定义了一个总体体系结构，并且下级系统遵循该体系结构。
- 在下级系统之间合理地注重重用以及经验共享。
- 对于要生产的工件以及下级和上级系统的工件之间的关系，有明确的理解。
- 定义了有效的变更管理策略并由所有团队遵循。

体系结构团队可以（但不总是）控制上级系统的开发。有关组织的更详尽的讨论，请参阅 [6]。

上级系统的生命周期

首先，您可以选择执行**业务工程**来更好的了解系统环境。如果是以下情况，这样做是增值的：

- 开发人员需要更好了解组织，
- 组织本身在业务执行方式上有不同种类，并且术语和流程需要统一，或者
- 软件工程工作是与业务再造工作协同进行的。

另请参阅 [6]。

此工作会产生业务用例模型和业务对象模型。或者，您可以选择执行有限的业务工程，只查看业务领域中的关键概念并将那些概念记录在业务对象模型中。这通常被称为领域建模。

一旦已“建立”了一组业务模型，您需要开始引出整个系统上的**需求**。如同其它系统一样，我们同样需要为互连系统组成的系统进行需求建模。用例模型是表达结果的一种十分普遍的方法，请参阅 [7]。查看此上级用例模型的最直接的方法是假定它完全捕获了系统的行为需求。不过，可能很少有这种情况。因为我们需要用其它系统来实现该系统，总体系统可能是相当复杂的。因此，试图在此级别过于详尽不是一个好主意。这样，上级用例模型通常会完整但简单地描述系统的功能需求。在此级别没有必要过于详细，因为详细的建模将在每个实现的下级系统中执行。在跨多个子系统的上级用例中，许多需求都不一定可见，通常也确实如此。这样的需求可以说对子系统是“局部”的。

分析与设计的目的是实现健壮的系统体系结构，这对于互连系统组成的系统自然是至关重要的。上级系统的开发人员必须实现健壮的下级系统结构，而完全不必为它们的内部结构而费心。因此，我们将使用子系统将系统的某部分构建成更小的部分。为了获得一组适当的子系统，并对如何在这些子系统上分配上级系统的职责有初步了解，我们开发了分析模型。当执行高级用例时，分析类应该表示由系统中的元素扮演的角色。因此，分析模型简要描述了完整的对象结构，类似于高级用例模型。

功能上相关的分析类一起分到子系统中。这样，我们获得了一个子系统结构，在某种意义上它是理想的（例如：它只基于功能条件），我们尚未考虑任何分发需求。通常有高度影响的因素是旧系统的存在。旧系统可以履行分析模型中定义的某项或多项职责。此类系统的存在甚至会导致重新划分分析中发现的职责，以便在最大程度上实现现有功能的重用。

设计结果可能是子系统结构与在分析期间基于功能条件定义的结构差别很大。因此最终以设计子系统的结构而告终，将由下级系统实现每个子系统，请参阅图 5。为了能够针对每个此类系统分别继续开发工作，要为每个子系统定义接口。事实上，定义接口是在上级级别执行的最重要的活动，因为接口提供了开发下级系统的规则。不定义设计类，唯一要做的事情是定义设计子系统的接口。

没有任何其它**实施**是作为上级系统的生命周期的一部分来执行的，除了可能有某项制作原型的工作来探索系统的特定技术方面。

最后的工作流程是**测试**，当组装不同的下级系统时，这种情况意味着集成测试，并且还测试协作中的互连系统是否按照规范执行每个上级用例。

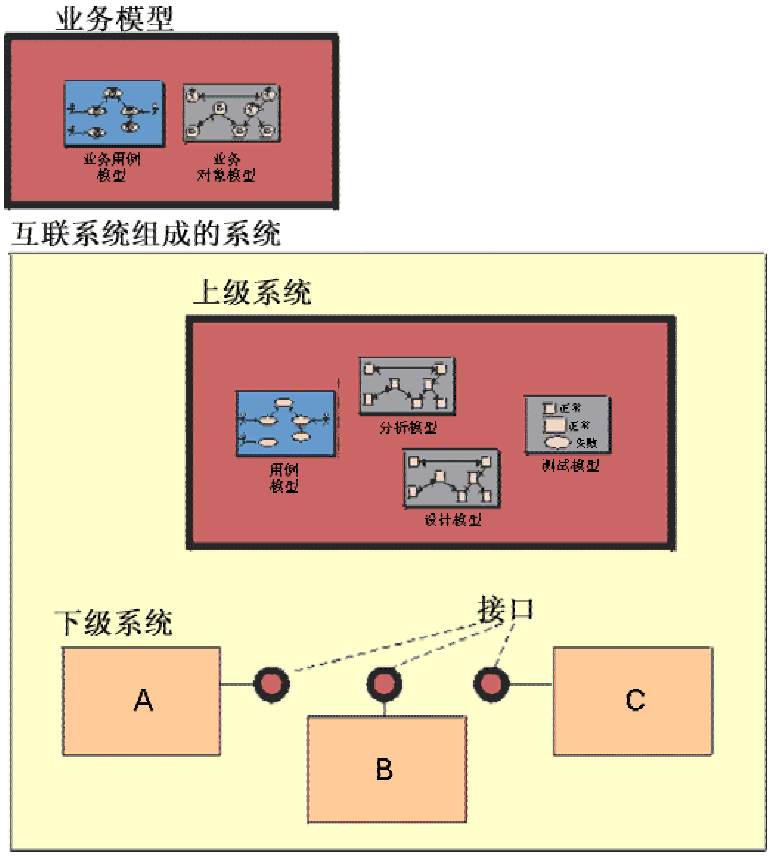

图 5. 上级系统是由一组模型描述的，其中，高级设计模型中定义的子系统将由下级系统实现。上级系统拥有下级系统的接口。

为了显示您将如何使用上级系统，这里是几个样本迭代计划；一个用于上级系统生命周期的先启阶段中的迭代，另一个用于精化阶段中的迭代。我们使用活动图来描述迭代计划。这些图中的操作状态与 Rational Unified Process 中定义的工作流程明细相对应。

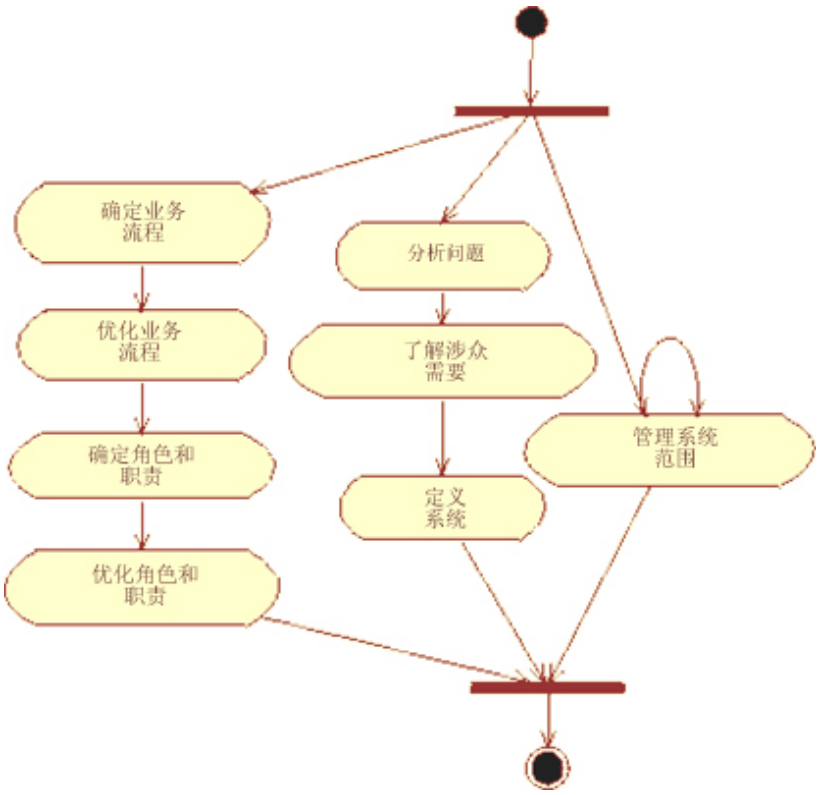

图 6. 描述上级系统的先启迭代计划示例的活动图。

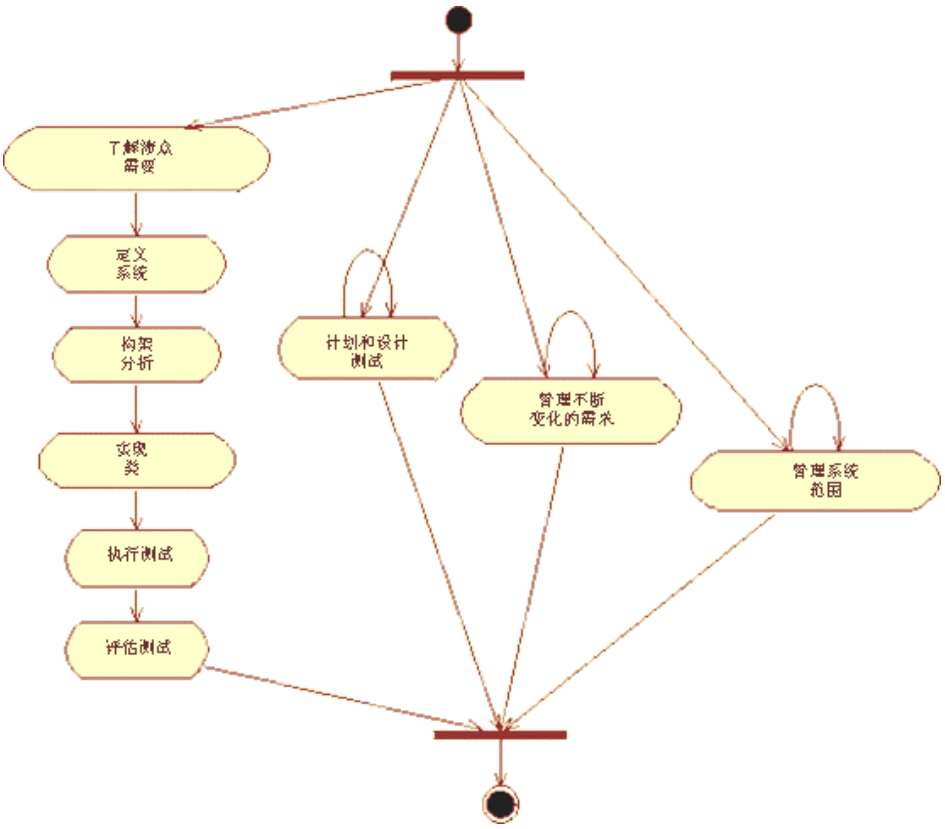

图 7. 描述上级系统的精化迭代计划示例的活动图。此处有操作状态“实施类”，因为可能要执行原型的有限实现来探索系统的技术方面。

下级系统的生命周期

以通常方式将每个下级系统作为黑匣来开发，认为与其通信的其它系统是参与者。如上所述，针对每个此类系统执行一组通常的活动并开发一组通常的模型。如果详尽定义了上级级别中的模型，则可获得不同级别的模型之间的完整递归，但如上所述，这实际上是很少存在的情况。

对于下级系统，将执行**需求**工作流程。上级系统的接口和用例将是您了解下级系统的边界及其参与者的主要信息来源。

当对下级系统执行**分析与设计**时，在上级系统中定义的接口将与高级用例一起成为您的“边界条件”。

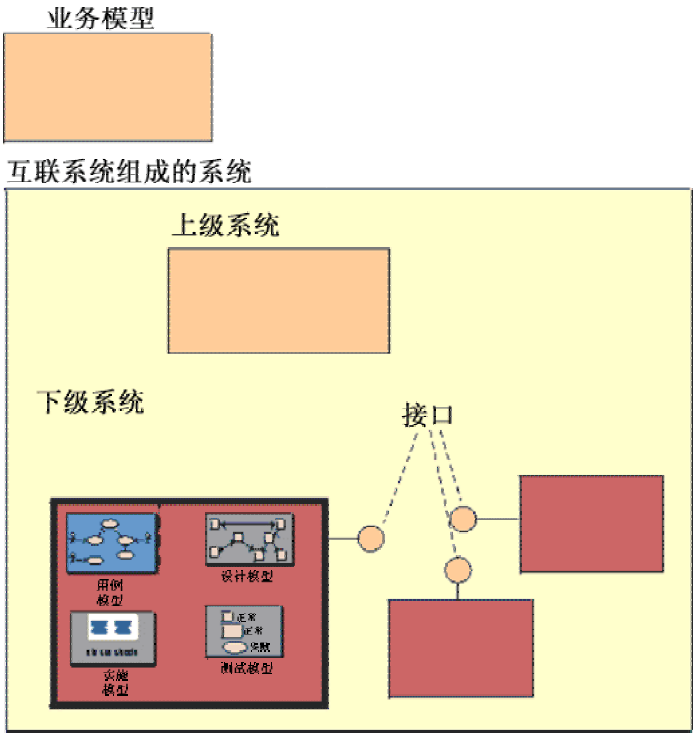

图 8. 下级系统由各自的模型集描述。

为了显示您将如何使用下级系统，这里是来自它的生命周期的两个样本迭代计划。

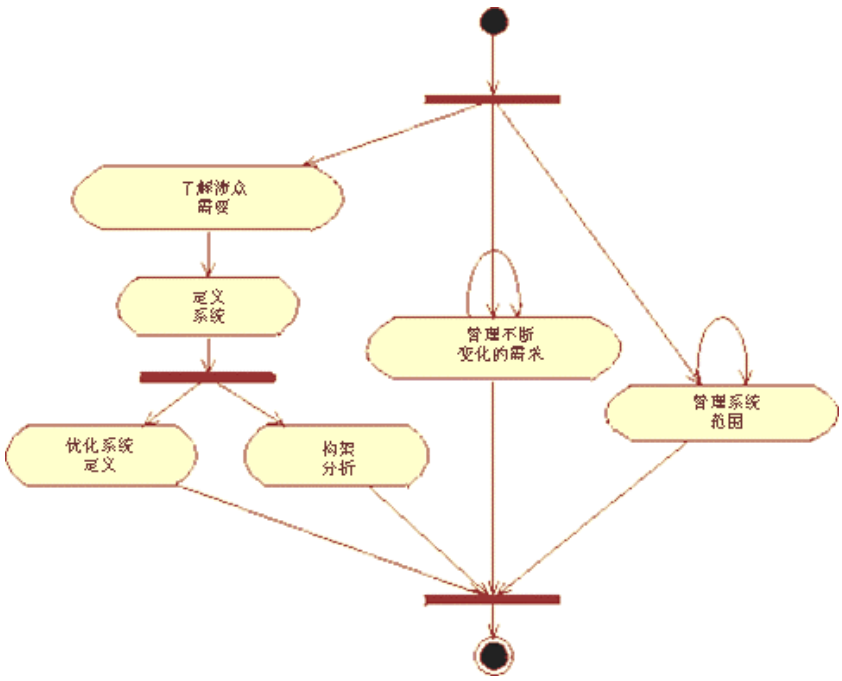

图 9. 下级系统的样本先启迭代计划。这是一个不完整的迭代，因为未产生可执行文件。

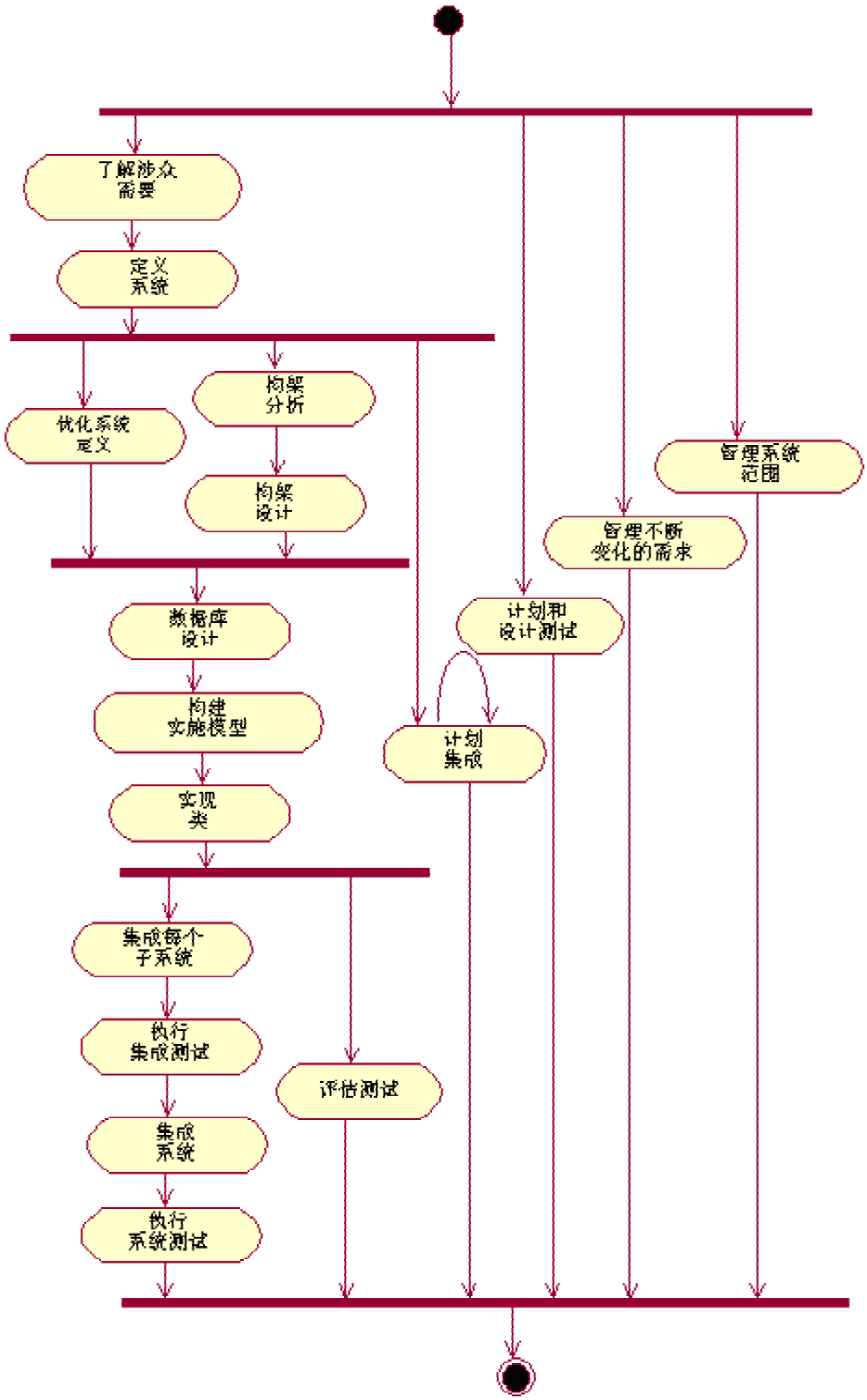

图 10. 下级系统的样本精化迭代计划。精化阶段的重点是完成优化的系统定义和体系结构。

互连系统组成的系统中的用例

应该为在互连系统组成的系统中的每个上级和下级系统建立一个用例模型。它们以下列方式相互依赖（另请参阅图 11）：

- （不总是，但通常）将上级系统中的高级用例分割到子系统中。每个“分割”都成为其下级系统的模型中的一个用例，请参阅图 11。
- 从一个下级系统的角度来看，其它下级系统是其用例模型中的参与者，请参阅图 12。

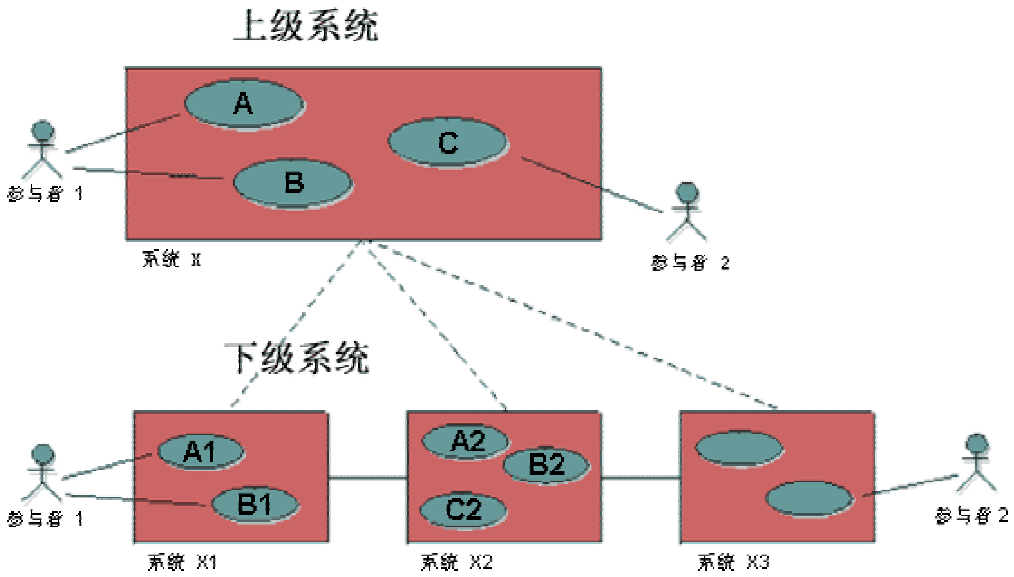

图 11. 上级系统中的高级用例和下级系统中的详细用例之间的关系。

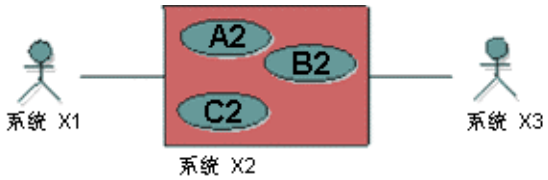

图 12. 在下级系统 X2 的用例模型中，其它下级系统 X1 和 X3 被视为参与者。

对于描述上级系统的用例，存在某些特殊的注意事项。由于在某种意义上，您将为每个下级系统重新描述所有需求，因此不必过于深入详细地描述这些用例。一般情况下，只需写下高级用例的事件流的分步概要通常就足够了，而不必以叙述性的文本详述。

在此用例模型中，不应该使用任何用例关系（泛化、扩展和包含）。通常，由于以下原因，这样做是不会增添价值的：

- 您将不会详细描述高级用例，因此，不必担心文本出现在多个位置。
- 当将高级用例“分割”到下级系统时，无论如何您都将构建信息。将其与其它构建机制混合起来可能会造成混淆。

这有一个重要的例外情况，即如果旨在查找互连系统组成的系统中的可重用组件时。构建上级用例模型来查找普通用例是查找可重用组件的有力方法。有关该主题的更多详细信息，请参阅 [6]。

互连系统组成的系统中的设计模型

互连系统组成的系统中的每个系统（包括上级和下级）应具有自己的设计模型。设计模型以下列方式相关：

- 上级系统的设计模型中的子系统定义下级系统的边界。
- 在上级系统的子系统上定义的操作是定义下级系统的接口的输入。

上级系统的设计模型没有下级设计模型描述得详细。您将会得出以下内容：

- 子系统，描述得很简短。
- 用例实现，根据子系统的协作方式。记录这些高级用例实现的通常方法是绘制序列图。通过产生这些图，定义将高级用例“分割”到下级系统，请参阅图 13。
- 子系统的操作。
- 子系统的接口定义。

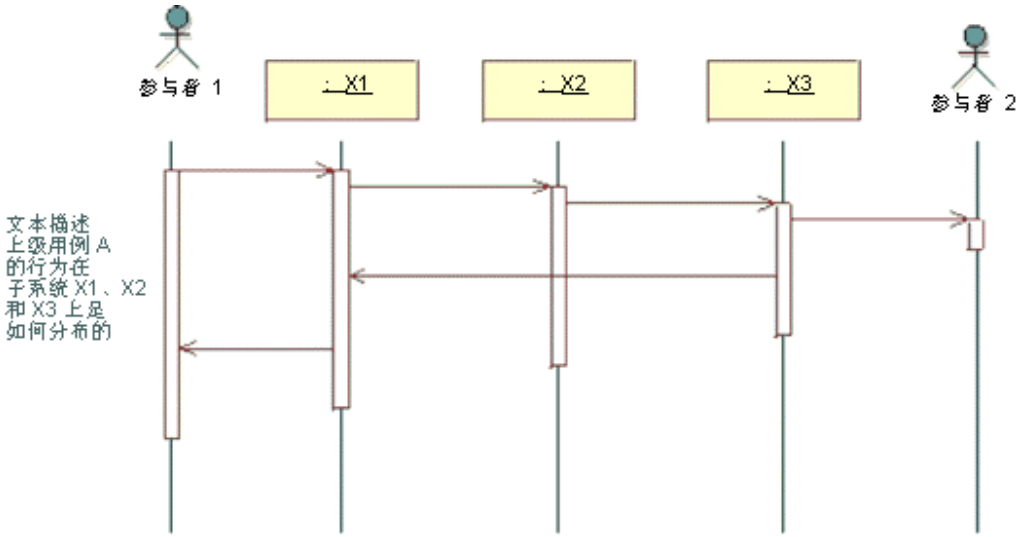

图 13. 上级用例 A 的实现的序列图。

互连系统组成的系统中的信息集

大多数组织耗费大量工作的区域是了解如何管理工件和正确了解它们的依赖关系。在前面的部分我们已明确讨论了上级与下级用例模型及设计模型之间的依赖关系。同时也存在需要考虑的常见依赖关系问题。

当系统经过生命周期的某个关口后，将产生可组织到信息集之中的工件 [8]，请参阅图 14。这些信息集是基于“一起”演进的工件而组织的。

- 您可以根据正在建立的应用程序类型来定制每个信息集的精确内容，但信息集仍保持相同。
- 您需要了解信息集之间的依赖关系，以便可以以有效的方式维护工件之间的可跟踪性。

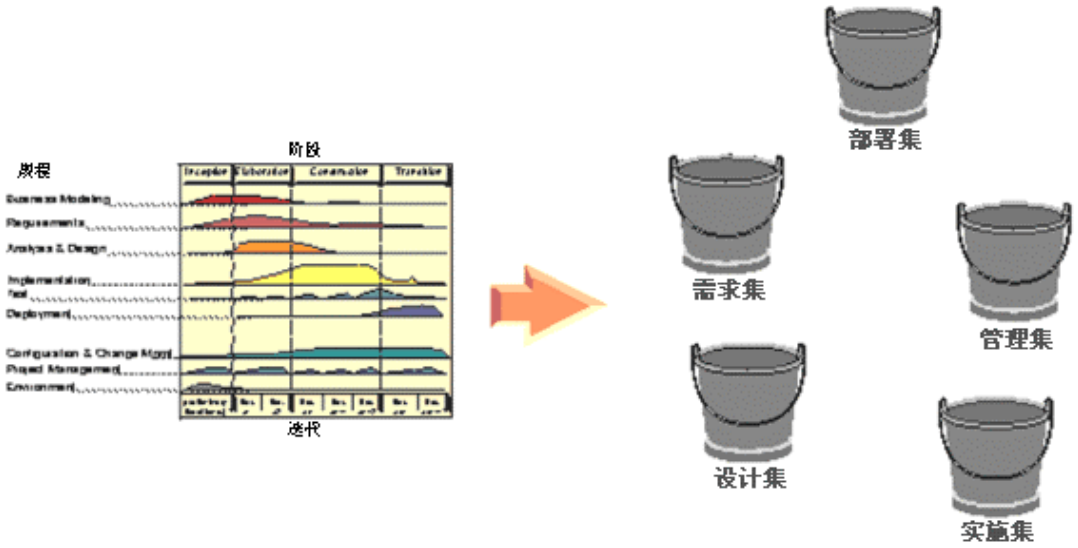

图 14. 系统的生命周期将产生信息集。

在互连系统组成的系统中，上级系统和每个下级系统将产生自己的一组信息集，请参阅图 15。

- 下级信息集与其相应的上级信息集具有依赖关系。
- 在下级系统之间的相应信息集中的内容类型可有所不同，因为应用程序的类型会有所不同。
- 除了相应的下级信息集实现在上级系统中定义的同一些子系统接口之外，这些下级信息集应当是独立的。

在维护上级系统和下级系统中的工件之间的可跟踪性上投入的工作量应保持在最少量。应当对维护系统内的可跟踪性划分优先级别。

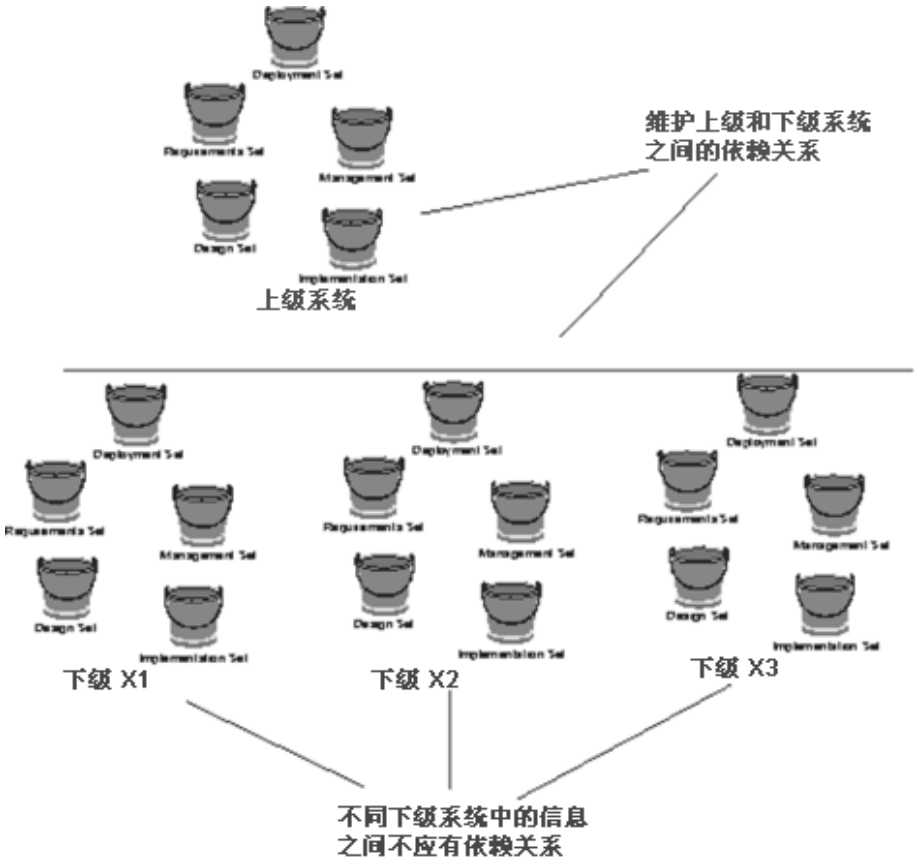

图 15. 互连系统组成的系统中的每个系统都将产生自己的一组信息集。

互连系统组成的系统中的体系结构

互连系统组成的系统中的每个系统（包括上级和下级）应定义了其体系结构。

对于上级系统，体系结构文档应讨论：

- 上级系统的关键用例或场景。
- 互连系统组成的系统的分层。
- 如何处理下级系统之间的重用以及重用什么内容。
- 关键机制和它们的实施，这些机制足够一般化，以至可用于所有下级系统。例如，所有下级系统应使用公用的机制进行通信、错误报告和故障管理，否则上级系统的行为方式将不遵循同质系统。

对于下级系统，体系结构文档应阐明：

- 互连系统组成的系统内的下级系统的角色。
- 下级系统的关键用例或场景。
- 下级系统将如何使用为互连系统组成的系统定义的分层结构。换一种说法，即您需要定义下级系统在上级系统的分层结构中将如何履行为其定义的角色。
- 将使用哪些一般的关键机制以及如何使用这些关键机制，并且将添加哪些特定于应用程序的关键机制。

- 将如何应用重用。具体来说，哪些子系统在两个或多个下级系统上是公共的，以及建立了哪些机制以使下级系统相互通信。

系统之间的关系

您已经了解到通常的系统开发活动也可应用于由互连系统组成的系统实现的系统。这是很有利的，因为它意味着您不必用与处理其它系统的方法有明显不同的方法来处理这样的系统。您也会从采用其它下级系统的形式的系统实施中很好地分离出上级系统。互连系统组成的系统中的每个系统都具有自己的生命周期。由于每个系统都可以有不同的特征，因此可以使用开发流程的各种变体来产生系统。根据 Rational Unified Process [2]，将为每个系统提供一个不同的开发案例。

互连系统组成的系统所涉及的系统之间的独立性的最后一点说明：

首先，请看下级系统。每个这样的系统实现上级系统的设计模型中的一个子系统。这些子系统依赖于彼此的接口，而不是显式地相互依赖，请参阅图 12。因此，您可以将一个子系统换成它的新版本，而不会影响其它子系统，只要新的子系统仍符合同一个接口。您在下级系统之间得到完全相同的关系。每个下级系统将它的环境视作一组接口。这意味着您可以将一个系统换成另一个系统，只要新系统对其它系统扮演相同的角色，即只要它可以被表示为同一组接口。各系统引用相互之间的接口（由上级模型中的子系统和接口之间的相应关系所指定）。

在下级系统的用例模型中，与其交互的其它下级系统的接口表示为参与者。可以说下级系统将另一个系统的接口看作是由相应参与者提供的，并因此从来不必直接引用其它系统，请参阅图 16。请注意，接口 B 出现在图 12 的多个位置中，表示它确实是由上级系统中的子系统和相应下级系统所引用的同一个接口。

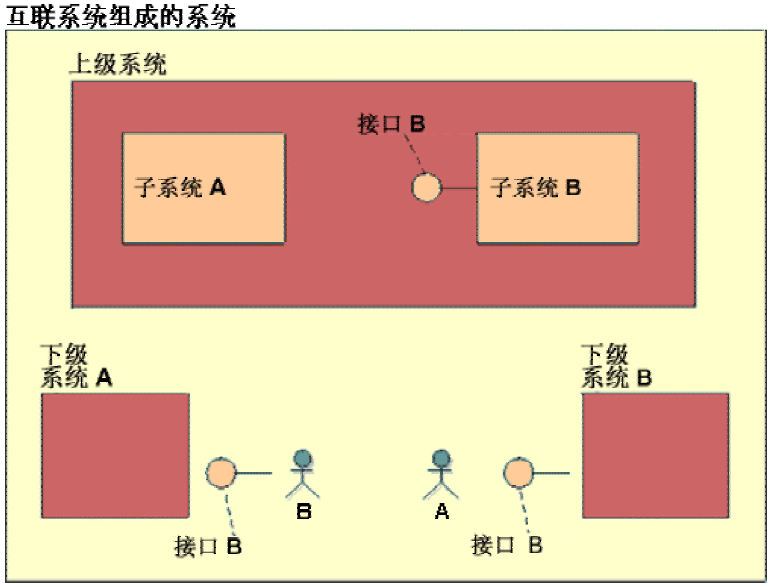

图 16. 上级系统的子系统只通过它们的接口相互依赖。因此，实现的下级系统获得了相同类型的独立性。在上级系统模型中，子系统 B 为其它子系统提供接口 B。相应的下级系统 B 因而需要为其它下级系统提供同一个接口 B。

那上级系统怎么样呢，它与其下级系统是什么关系？就以下意义而言，它与它的实施系统无关：每个这样的系统只是上级系统模型中所指定内容的实施，它不是其规范的一部分。出于实际的原因，必须定义不同级别的系统之间的可跟踪性链接，以便跟踪需求，执行此操作的最“精简”的方法是只在接口之间定义此类链接，请参阅图 11。事实上，甚至可以说下级系统只是一些实施，这些实施提供在上级模型中定义的接口。

但是，这对于不只是简单示例的系统是不够的。接口只指定了在特定交互点要进行的操作。一个下级系统可以有数百个接口，并且每个接口有数十个操作。在接口描述中，将某个接口处的输入与另一个接口处的一个或多个输出相关是不切实际的做法。这就是为什么您需要用例来解释下级系统的语义。

您可以得出以下结论：当某个系统是由互连系统组成的系统实现时，所涉及的每个系统都是独立于其它系统的，但它们强烈依赖于相互之间的接口。对于并行开发下级系统，这为您提供了一个非常好的平台。

应用程序区域

互连系统组成的系统的体系结构和建模技术可用于不同类型的系统，例如：

- 分布式系统
- 非常大型或复杂的系统
- 多个业务区域相结合的系统
- 重用其它系统的系统
- 系统的分布式开发

也可能是相反的情况：从一组已经存在的系统，通过组装这些系统来定义互连系统组成的系统。事实上，在某些情况下，这是大型系统在演进早期阶段的演进方式。您意识到有一些可以互连的系统，然后将它们连接起来，因而创建了一个“大型系统”，它比两个单独的系统更有价值。

事实上，对于任何系统，若将该系统的不同部分视作各部分自己的系统是可能的，则建议将其定义为互连系统组成的系统。即使目前它是单个系统，以后可以证明将该系统分割成多个独立产品是必要的，原因是由于分布式开发、重用原因或客户只需要购买系统的某些部分（需要提及一些示例）。

作为结论，我们将细看几个案例，在此可使用互连系统组成的系统的体系结构。对于每个示例，我们将显示，正在讨论的系统必须被视为单个系统和一组独立的系统，表明它应被视为由互连系统组成的系统实现的上级系统。

大规模系统

电话网络也许是世界上最大的互连系统组成的系统。这是一个极佳的示例，其中需要多于两个的系统级别来管理复杂性。它也是以下情况的示例：最上级系统由标准化团体所有，而不同的竞争公司开发必须符合此标准的一个或多个下级系统。在此，我们将讨论移动电话网络 GSM（全球移动电话系统），来显示将大规模系统实现为互连系统组成的系统的优势。

非常大型的系统的功能通常结合了多个业务区域。例如， GSM 标准覆盖了整个系统，从呼叫用户到被叫用户。换言之，它包括移动电话和网络节点的行为。因为系统的不同部分是单独（甚至由不同类型的客户）购买的各部分的产品，它们应被视为各自的系统。例如，开发完整的 GSM 系统的公司将向用户销售移动电话，向电话接线员销售网络节点。这是将 GSM 系统的不同部分视为不同下级系统的一个原因。另一个原因是将像 GSM 这样大型和复杂的系统作为单个系统开发要花太长的时间；必须由多个开发团队并行开发不同的部分。

另一方面，由于 GSM 标准覆盖整个系统，将系统视为一个整体（即上级系统）也是有道理的。这将有助于开发人员了解问题领域以及不同部分是如何彼此相关的。

分布式系统

对于分布在多个计算机系统上的系统，互连系统组成的系统的体系结构是十分适合的。按定义，分布式系统始终由至少两部分组成。因为在分布式系统中良好定义的接口是很必要的，所以这些系统也非常适合以分布式方式*开发*，即由多个自治开发团队并行工作。分布式系统的下级系统甚至可作为各自的产品进行销售。因此，将分布式系统视作一组单独的系统是很正常的。

分布式系统的需求通常包含整个系统的功能，并且有时不同部分之间的接口尚未预定义。而且，如果对于开发人员来说，问题领域是新的，则他们首先必须考虑整个系统的功能，而不管它将如何分布。这些是将它视为单个系统的两个非常重要的原因。

旧系统的重用

几乎在大多数情况下，大型系统会重用旧的系统。可将旧系统描述成下级系统。然后您会为旧的系统“重建”用例模型并且可能重建分析模型，以了解在上级系统的更大型的环境中它可以如何工作。这些重建的模型不必一定是完整

的，但至少它们需要包含旧系统的功能（对互连系统组成的系统的其余部分的功能有直接影响，或可能需要修改）。

预制包的使用

一个系统可以是两个或多个预制包的集成和定制。一个好的示例是企业资源计划（ERP）系统。许多 ERP 系统组装了多个下级系统，如 MRP（物资资源计划）、库存管理和供应链管理等。类似的组装可用于其它领域，如人力资源或薪资应用程序。它们很像预制系统，您必须专门研究并互连其它标准包，以便获得完整的系统。了解这组包需要为上级系统一起执行的操作。这种情况是在财务团体中的许多客户目前所面临的。

摘要

本白皮书简介了互连系统组成的系统的体系结构模式。此构造不仅允许在一个模型中递归，它将每个子系统视为一个合格的系统并允许在每个系统的所有工件集之间进行递归。简介的体系结构用于由多个通信系统实现的系统。每个涉及的系统用它自己的一组模型（独立于其它系统模型）来描述。

使用此项技术的优势是显而易见的：您可以接近相当复杂的问题并使用“分而治之”技术来理解这些问题。但缺点是您面临着更多开销和跟不上进度安排的风险。我们也看到了以下示例：组织发现很难对上级系统使用迭代生命周期，因而面临着风险被延续到上级系统的生命周期结束的风险。您也需要密切注意是否遵循了合理和有效的重用策略，以避免开发一组“卡壳”系统。

给出的示例阐明了为互连系统组成的系统建模的体系结构在多个不同的应用区域都是很有用的。事实上，可以对任何系统使用建议的体系结构，只要将系统的不同部分视为各自的系统是可能的。

参考资料

[1] Jacobson, I.; Palmkvist, K.; and Dyrhage, S., *Systems of Interconnected Systems*, ROAD, 2(1), 1995.

[2] *Rational Unified Process* V5.1.

[3] Rumbaugh, J.; Booch, G.; Jacobson,I., *UML Reference Manual*, Addison Wesley Longman, 1999.

[4] Herbert A. Simon, *The Sciences of the Artificial*, MIT Press, 1981.

[5] Jacobson, I.; Bylund, S.; Jonsson, P., *Using Contracts and Use Cases to Build Plugable Architectures*, Journal of Object-Oriented Programming, May/June, 1995.

[6] Jacobson, J.; Griss, M.; Jonsson, P., *Software Reuse – Architecture, Process and Organization for Business Success*, Addison Wesley Longman, 1997.

[7] Jacobson, I., *Use Cases in Large-Scale Systems*, ROAD, 1(6), 1995.

Rational®
the software development company

Dual Headquarters:
Rational Software
18880 Homestead Road
Cupertino, CA 95014
Tel: (408) 863-9900

Rational Software
20 Maguire Road
Lexington, MA 02421
Tel: (781) 676-2400

Toll-free: (800) 728-1212
E-mail: info@rational.com
Web: www.rational.com
International Locations: www.rational.com/worldwide