第3章 背景知识

本章将简要介绍教程中用到的一些重要概念，并提供相关参考资料，让读者获得较全面的背景知识。读者如果已经具有计算机安全和 UML 方面的知识，将会很有帮助，因为本书省略了有关知识较深入的部分。我们在后面会提供一些建议阅读的背景资料。当然，有许多更好的相关资料在书中没有提及。在此，我们只是简单地回顾本书中涉及的主要概念。

3.1 安全工程

我们先描述在本书中起重要作用的计算机安全工程问题。相关背景知识可参阅教材[Gol99，And01]，以及[VM02][①](关注软件)。此外，[APG95]包含有关方面一些很有意义的文章；[SS75，Gas88，AN96]等是安全工程方面的一些“经典”参考书；密码学的介绍可参阅[MvOV96，GB99]等；[ERW81]是一本关于数据库安全性和完整性方面的经典书籍。

通过开放式、未受保护的网络进行通信常常容易受到攻击。比如在 Internet 中，阅读或删除交换的消息相当容易，插入其他消息也很容易。在无线网中，至少阅读或插入消息会更加容易，因为更容易接近这种网络。因此，在不可信网络上的安全通信需要特定的机制，如加密技术和密码协议。密码协议是对信息交换的一种描述，其中包括加密数据，以建立协议参与方之间的一种安全关系，如一条安全信道。正如简介中所说，密码协议很难设计，而且很容易出现非常不明显的错误。

除了攻击分布式系统中的通信连接外，在物理访问可能的情况下，一个敌手也可能试着直接攻击物理系统节点。如果时间充裕，获得对缺乏特殊保护的系统组件的数据访问会非常容易，获得对其数据或行为的操作也会非常容易。在系统中，敌手可能会拥有一个安全关键节点(例如，如果敌手可以是系统用户的话)。因此通常要求系统使用特定受保护的硬件，比如智能卡，目的是实现防篡改。

本书基于模型的安全工程方法已经被设计适用于面向对象和面向组件的系统。尤其，面向对象非常适合作为安全系统的概念基础，因为系统设有一种通用机制来控制数据的访问，即方法调用；系统提供信息隐藏，即通过在对象中封装数据；封装入一个对象中的数据仅能 通过对象来访问，消息是唯一通信的方式等。但要注意的是，为了提供保护，在面向对象

① 对于德语读者，推荐参考[Eck03]。

的执行环境中对这些机制的实现必须能够经受各种动机的攻击，使得它们不能被绕过。关于这个方面更多的信息请参见[Gol99，第 17 章]。

一个安全策略概括的是对一个系统的保护需求。接下来，我们将介绍本书教程中涉及的一些重要安全需求和概念。

公平交易

当进行电子商品交易时，公平交易需求强调该交易能够在防止参与方欺骗的方式下进行。例如，买方必须预付货款时，他应该能够证明自己已经成功支付，并且能够在最终没有收到商品的情况下成功要回预付货款。

不可抵赖

提供公平交易的一种方式是使用对某些操作的不可抵赖性安全需求，即这个操作事后是绝对不可能被否认的。换句话说，这个操作通常依据某个可信第三方是可以得到证明的。

基于角色的访问控制

控制受保护资源访问的一种重要机制是基于角色的访问控制的概念。为了使许可权易于管理，尤其是在具有大量或频繁变更用户基的系统中，许可权并不直接分配给用户，而是让用户拥有一个或多个与他们在组织机构中与职责相关联的角色，然后将许可权分配给相关角色。

安全通信连接

系统中两个不同部分之间的敏感性通信需要受到保护。在这里，一个安全通信连接的相关需求被假定为能够对传输的数据提供机密性和完整性保护。

机密性和完整性

两个主要的数据安全需求是拥有机密性（或秘密性）和完整性。数据的机密性，即数据只能被合法的用户读取。数据的完整性，即数据仅能被合法的用户修改。

真实性

这是第三种主要的数据安全需求，它的变种有很多。其中重要的两个是消息真实性和实体真实性。消息真实性（或数据源真实性），即可追踪某些数据片段的真实来源。实体真实性，即保证能够识别协议参与方，尤其能够保证参与方当时确实是主动参与了该协议。提供真实性的过程叫做鉴别。

新鲜性

一个消息是新鲜的，即如果它在系统当前执行期间被创建（例如，在当前协议执行期间），则不可能是来自敌手的一条旧消息的重放。nonce 是一个随机值，只能够使用一次，并因此而得名。比如，建立包含一个最新创建 nonce 的某消息，构造的消息本身就是新鲜的。

安全信息流

确保计算机系统安全性的一种传统方式是设计多级安全系统。在这种系统中，有不同

的数据敏感级。为了简化，人们常常考虑设置两个安全级：高级别，意味着高度敏感或高度可信；低级别，意味着不太敏感或不太可信。当一个系统的可信部分与不可信部分交互时，要确保从可信部分到不可信部分不存在间接的敏感信息泄露。为此，需要实施“不可向下流”策略：低级别数据能够影响高级别数据，但是反过来不行。与此相对，“不可向上流”要求系统的不可信方不能够间接操作高级别数据：高级别数据可以影响低级别数据，但是反过来不行。这些安全需求，称为安全信息流或者无干扰性 [GM84]，是机密性和完整性的相当严格的定义，能够检测出被称作隐蔽通道的潜在信息流。

受监护的访问

有一种主要的安全机制是访问控制，它可以确保只有合法用户才能访问系统安全相关部分。通常，访问控制的实施是通过监护方：比如，在 Java 安全体系结构中，监护对象控制对受保护对象的访问；类似地，在 CORBA 系统中，访问决策对象控制对受保护对象的访问。

3.2 统一建模语言(Unified Modeling Language)

统一建模语言(UML)[RJB99]是用于描述面向对象软件系统的实际行业标准，它同样适用于面向组件的系统。UML 是一种图形化语言，可以从结构和行为等方面对软件进行描述。关于 UML 的介绍可参考[SP99，Fow04]①等。这里主要介绍 UML1.5 [UML 03]②。

UML 图对一个系统设计的不同层面进行多种视图描述。其中有一些 UML 图可以对系统不同抽象层面的不同方面进行描述，在本书中，我们将用到以下几种 UML 图。

- 用例图：通过描述用户与系统的典型交互，对系统提供的功能进行抽象视图描述。用例图常常是在系统设计之前，为了与客户进行协商，采用的一种非正式的模式。
- 类图：定义系统的静态类结构，即类(包括属性、操作和信号)，以及类之间关系。在实例层面上，相对应的图叫做对象图。
- 状态图：描述一个独立对象或组件的动态行为，事件可以引发状态的变化或动作的执行。它是对 Harel 在[HG97]中描述的状态图的一种改版。
- 顺序图：描述对象或系统组件之间传递消息(特别是方法调用)的交互过程。
- 活动图：描述系统中若干组件之间的控制流，通常处于比状态图和顺序图更高的抽象层次上。它可以用于将对象和组件放在整个系统行为的具体环境中，或者用于更详细地解释用例。
- 部署图：描述系统软件组件到系统硬件物理结构的映射。
- 子系统(一种特殊的包)：集成不同类型的图之间和系统规范的不同部分之间的信息。

---

① 对于德语读者，我们同样推荐文献[Bre01，Rum04]。

② 未来版本 UML2.0 中的一些符号参见附录 A。

除了顺序图，还有协作图，两者表示的是类似的内容。此外，还有组件图，表示的是部署图中包含的部分信息。为了简化表示，这里省去了这两种图，在我们的方法中也可以使用它们。

对每一种图，我们只需要其各标记元素中比较简单的片段。在以下各节中，我们只会粗略地描述本书中涉及的上面这几种类型图的特性。尽管还有许多其他类型图的元素可以在我们的方法中应用，但是在我们的表示中它们并不是必需的。

3.2.1　用例图(Use Case Diagrams)

用例图能够以一种抽象的方式构建一个系统的功能，以及描绘一个系统和一个用户之间的交互。用例图包括用例(use case)和参与者(actor)两个元素。一个用例是一组连贯的交互，这组交互属于特定的系统应用。一个参与者拥有一个名称，定义了实体的角色集。实体可以是人或者是其他的计算机系统。一个人的角色可以是客户(Customer)，如图 3.1 所示。从一个参与者到一个用例之间的连接，表示该用例所属的系统可以对参与者(actor)所代表角色的任何实体提供用例(use case)所表示的服务。目的是当所描述系统的服务被使用时，用例的实例和参与者的实例之间交互。使用其他类型的图，如活动图或顺序图，目的是对这个交互进行更加详细的描述。在这些类型的图中，还有更多的模型元素，这些模型元素也可以在我们的方法中应用，如扩展(extends)和包含(includes)关系等。

图 3.1　用例图

图 3.1 是一个用例图实例。其中“顾客”(Customer)(参与者)，要执行“购买商品”(buys good)(用例)，“商家”(business)(参与者)，要执行“销售商品”(sells good)(用例)。

3.2.2　类图(Class Diagrams)

对象(object)是“具有定义明确的边界和标识的实体，它封装了状态和行为，其中状态由属性和关联描述，行为由操作、方法和状态机描述。对象是类的实例(instance)”[UML03，p. Glos.-10]。类(class)是“对一组拥有相同属性、操作、方法、关联和语义的对象的描述，类可以使用一组接口来规范可提供给其环境的一系列操作”[UML03，p. Glos.-4]。我们使用类图描述各个类和它们在系统中可供使用的接口，以及类之间的关联，如依赖关系。当一个模型元素依赖于另外一个模型元素时，后者的改变可能会影响前者。

在这些图形符号中，一个被分成三部分的矩形代表一个类，这三个部分分别给出这个类的名称、属性和操作。对象使用同样的方法表示，区别在于对象的名称加了下划线。这种用来区分类型和实例的符号机制，同样适用于其他类型的图。带虚线的开放箭头用来表示类与类之间的依赖关系。接口使用接口规范表示，图形符号是一个贴有≪Interface≫标签、包

含该接口提供的操作和信号的矩形，带虚线的中空箭头用来表示其来自实现该接口的类。在 UML 图中，符号≪ ≫中的标签集被称为构造型(stereotypes)，参见第 3.2.8 节中对 UML 构造型概念的解释。为了简化，可以去掉接口规范，而用一个附在类矩形上的圆圈来取代。一个带有构造型≪call≫(或者≪send≫)的依赖关系箭头从类 dep 指向类 indep，表示类 dep 的实例可以调用类 indep 实例的方法(或者给后者发送信号)。特别地，类 dep 的实例知道类 indep 的实例。如果箭头指向类 indep 的一个接口，dep 只能调用接口规范中罗列的操作或者发送接口规范中罗列的信号。例如，在图 3.2 中，发送者(Sender)可以发送带有参数 d 的信号 transmit 给接收者(Receiver)，但是通过接口 receiving 访问接收者(Receiver)的对象只能调用不带有参数的 receive 操作，并获得一个 Data 类型的返回值。

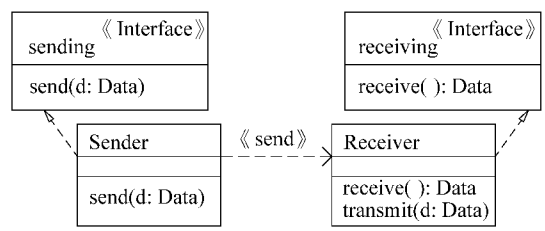

图 3.2 类图

依照惯例，我们通过给属性类型加下划线的方式区分常量属性。我们进一步规定常量属性使用它们的值命名。因此我们可以省略明确赋予常量属性初始值。例如，K:**Keys** 表示一个值为 K 的常量，$K \in$ **Keys**。

3.2.3 状态图(Statechart Diagrams)

UML 状态图用来描述状态机，该状态机描述一个实体(如对象或组件)经由事件触发，以及触发后的行为的状态序列[UML03]。它们源自 Harel 在[HG97]中所描述的状态图，包括状态和状态之间的转换。

状态(state)是“对象生命周期中的一种情况或状况，它满足某条件，执行某行为或等待某事件”([UML03, p. Glos.-14])。状态使用包含该状态名称的方形表示。在状态的入口(出口)处包含入口(出口)动作。一个状态可以被划分为顺序(并发)子状态，之后被称为顺序复合状态(并发复合状态)。如果一个顺序复合状态是激活的，则其中的一个顺序子状态也是激活的；如果一个并发复合状态是激活的，则其所有的并发子状态都是激活的。一个状态图和它的子图各自包含一个初始状态，可能包含一个或多个终止状态，初始状态使用一个实体圆表示，而终止状态用一个包含小实心圆的圆形表示。如果不存在终止动作的话，也可以没有终止状态，如图 3.3 所示。

一个带有标签 e[g]/a 的转换(transition)表示：处于第一个状态的对象将执行动作 a，并且在事件 e 发生和监护条件 g 满足时进入目标状态，即转换发生了。监护条件可以是一个逻辑公式，包含如数据表达式之间的＝(等号)、≠(不等号)，以及逻辑连接符号，如 ∧

(与)、∨(或)、¬(非)和⇒(导出)。动作(action)可以调用一个操作或发送一个信号,记为 call($op(args)$)或 send($sig(args)$),也可以给一个属性赋值,记为 $att:=val$。当进行方法调用和信号发送时,关键字 call()和 send()在图中常常被省略,以增加可读性;而只是使用方法或信号的名称以及参数来替代。一般地,发送给对象 *obj* 的消息使用名称 *obj* 作为消息名称的前缀;如果没有产生混淆的话,这也可以被省略。拥有相同源对象和目标对象的转换是内部的(internal),这意味着它们可以进行转换但不需要调用进入或退出动作,并且只要状态处于激活状态,内部活动就一直被执行。内部转换的实质是模型化一个不会改变对象状态的事件的响应。

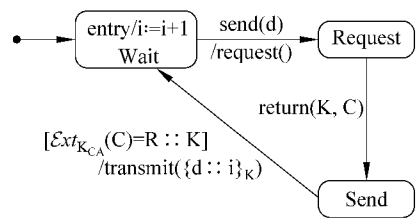

图 3.3　状态图

图 3.3 给出了一个简单的例子。该状态图包含三个状态,分别是 Wait、Request 和 Send(没有子状态、动作或活动)以及一个初始状态。初始执行时,首先进入 Wait 状态,并且执行入口动作:属性 i:=i+1,当消息 send 到达时,参数存储在变量 *d* 中,发送消息 request,进入 Request 状态。随后,当接收到 return 消息时,它的两个数值存储在变量 K 和 C 中,进入 Send 状态。然后,当条件 $\mathcal{E}xt_{K_{CA}}(C)=R::K$ 满足,则发送消息 transmit $(\{d::i\}_K)$,重新进入 Wait 状态。①

3.2.4　顺序图(Sequence Diagrams)

顺序图描述对象间按照时间顺序进行的交互。例如,它描述参与交互的对象和对象间交互的消息列。一个顺序图可以以一般形式存在(描述所有可能的交互情景),也可以以实例形式存在(描述一个实际的交互情景)[UML03, p. Glos.-13]。本质上,一个顺序图对一组对象或组件之间的交互进行说明,而这些组件或对象的名称在顺序图的第一行给出。每个名称之下有一条垂直的竖线,叫做生命线(life line)。当对象处于激活状态时,生命线上使用窄长方形代替竖线来表示顺序图中这段时期。这里的带线箭头被称作连接(connection),连接附带的消息表示该消息从带线箭头发出的对象生命线发送到箭头指向的对象。为了增强可读性,发送到对象 obj 的消息名称的前缀 obj 可以被省略,因为它在顺序图中是唯一的。对顺序图中的各个方法 *msg* 和各个整数 n,msg_n 代表顺序图中最近被接收到的、调用 *msg* 操作的第 n 个参数。在这里我们并不简单地使用那些表现为消息参数的表达式,因为攻击者可能修改传递的内容。注意在状态图中这点可以被自动实现,因为不同的状态图拥有不同的命名空间。为了增强可读性,我们允许在图中定义句法简写式,形式是 $var:=exp$,其中 *var* 是不带有其他使用目的的局部变量且表达式 *exp* 中不包含 *var*。在给顺序图指定语义之前,变量 *var* 在所有场合都应该由表达式 *exp* 代替。例如在图 3.4 中②,k 作为密码表达式 **fst**($\mathcal{E}xt_{K''}(\mathcal{D}ec_{K_C^{-1}}(c_k))$)的简写式。图中也可有写在方括号[ ]中的同带

① 注意该密码表达式与本节内容无关,且将会在第 3.3.3 节中进行解释。

② 同样,该密码表达式与本节内容无关,且将会在第 3.3.3 节中进行解释。

线箭头相关联的条件，当图被执行时，必须实现这些条件，否则相关箭头上的操作将无法执行。注意消息的发送者或接收者可能不包含在顺序图中，在这种情况下，带线箭头从图内指向图外或者从图外指向图内。

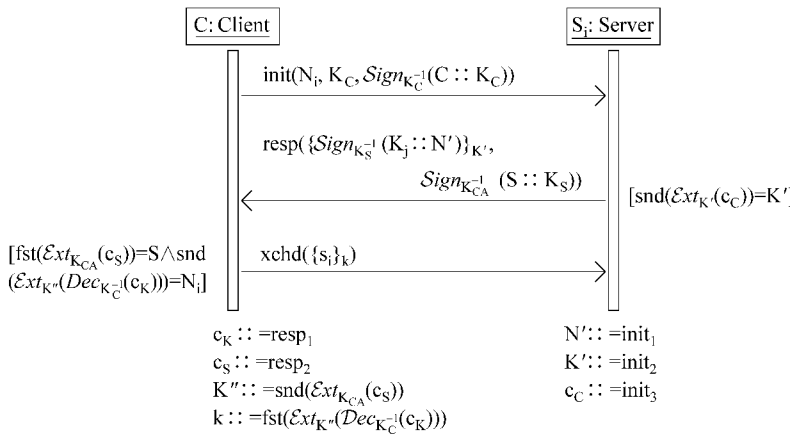

图 3.4 顺序图

图 3.4 给出了一个顺序图实例。该顺序图规范了两个持续处于活跃状态的对象之间的安全协议，C 和 S_i 分别是 Client 类和 Server 类的一个实例。该协议的开始是由客户端 C 向服务端 S_i 发送带有三个初始参数 N_i、K_C 和 $\mathcal{S}ign_{K_C^{-1}}$(C:: K_C)的消息。如果满足条件 snd($\mathcal{E}xt_{K'}$(c_C))=K'，其中 c_C 是 $init_3$ 的简写式，则客户端 S_i 随后向客户端 C 返回带有参数 {$\mathcal{S}ign_{K_S^{-1}}$($k_j$:: N′)}$_{K'}$，$\mathcal{S}ign_{K_{CA}^{-1}}$(S:: K_S)的消息 resp。客户端再检查顺序图左边的条件，如果有效，则客户端向服务端返回消息 xchd，协议结束。否则，如果条件不能完全满足，则该图的执行在相关断点停止。该协议将在第 5.2 节详细解释。

3.2.5 活动图(Activity Diagrams)

在 UML1.x 中，活动图是状态图的一个特例，状态用来对包含一个或多个对象或组件的程序过程建模，活动图则对这些对象或组件的执行进行协调[UML03, p. 3-156]①。在活动图中，并发组合状态使用同步条(synchronization bar)表示，因此对一个并发组合状态 S，从 S 的并发子状态的初始状态出发的转换由一个从同步条出发的转换代替，且忽略用来描绘并发状态的线条。类似地，进入 S 的并发子状态的终止状态的转换由进入一个同步条的转换代替。

活动图可以划分为多个泳道(swimlane)，每个泳道包含其对象(及其所属类)的名称，或者组件(组件行为由泳道中的活动来建模)的名称。我们假定这种划分的定义是良好的，即泳道中标有组件 C 的活动只能访问 C 中的数据。为了增强可读性，我们省略了属性名称前缀——对象的名称，因为它们已经被作为泳道的标识给出。

活动图的状态可以分为下几种类型。一种状态：它没有内部活动、内部转换、退出行

① 未来版本 UML2.0 的改变见附录 A。

为，或对外非完全转换，但至少有一个对外完全转换，这种状态称为行为状态(action state)[UML03, p. 2-171]。另一种状态：它的内部活动建模成一组拥有一定持续期的、非原子性的顺序步骤，这种状态称为子活动状态(subactivity state)[UML03, p. 2-174]。行为状态和子活动状态由顶部和底部为直线、两边为突出曲边的盒形表示。

图 3.5 中给出了一个活动图的实例。在该图中，三个对象 C、L 和 I 并发执行。对 C，活动 c 重复执行直到计数器 nt 达到值 *limit*。类似地，对 L，活动 l 重复执行直到计数器 n 达到值 *limit*，对 I，活动 i 只是无限重复执行。

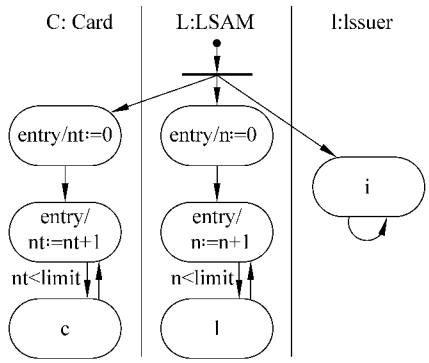

图 3.5　活动图

3.2.6　部署图(Deployment Diagrams)

部署图是一种“描述运行时的处理节点，以及节点中的组件、进程和对象配置的图”[UML03, p. Glos.-6]。这些节点由盒形图表示，可以由代表通信链接的实线连接。一个节点可以包括若干组件，由一个左边插入了两个小矩形的大矩形来表示。和类图中一样，组件可以由带虚线的箭头(表示通信依赖关系)链接。它们可以拥有接口，并且可以包含类或子系统模型。因此，链接(link)代表系统中不同节点之间的物理通信链路，而依赖关系(dependency)描述的是不同组件之间的逻辑连接关系。

图 3.6 给出了一个部署图的例子。该图中包含两个节点——客户端(client machine)和服务器(server machine)：客户端(client machine)包含一个带有接口 get_password 的组件 client apps，而接口 get_password 中又包含一个对象 browser；服务器(server machine)包含一个组件 Web server，该组件包含一个对象 access control。两个节点由构造型≪Internet≫标记的链接连接。还有一个从组件 Web server 到 get_password 的依赖，由构造型≪secrecy≫标记。后一个构造型已经包含在第 4.1.2 节定义的 UMLsec 扩展中。因此，为了获取密码，Web server 被规范为能够与 browser 通信，并且这可以通过一个 Internet 连接做到。

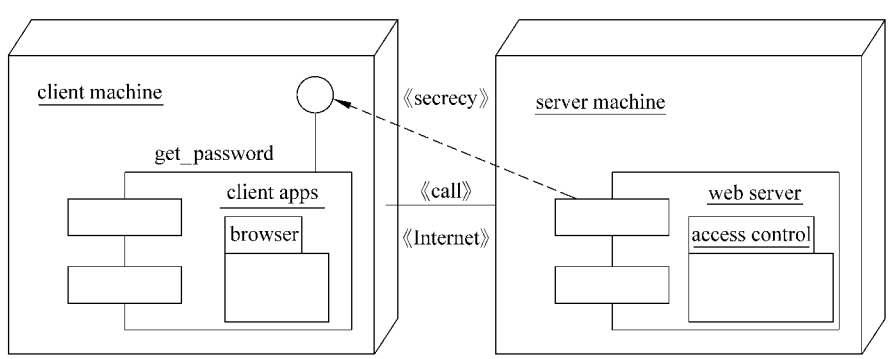

图 3.6　部署图

3.2.7 子系统(Subsystems)

包(package)是简化 UML 图表示的一种标记方法。人们可以将由图描述的模型中的一些部分组合在一个包中。因此,只有包标记,而不是其代表的图组,要在整体图中出现。

在此,我们采用一种特殊的包——子系统,即多个模型元素的组合,模型元素表示实际系统中的行为单元[UML03, p. Glos.-15]。一个对整个系统进行建模而不是对系统某部分建模的子系统,就称为一个系统(system)。子系统可以有接口,并且可以在所谓的静态结构图(static structure diagram)中使用依赖关系来同其他的子系统连接。在子系统的左上角,人们可以说明那些可以从子系统外部调用的操作。在一个子系统中,人们可以将规范元素和实现元素明确区别;刻画这些区别不是必需的,且不在本书考虑范围。

图 3.7 给出了一个子系统的实例。像类图中的描述一样,Channel 子系统包含两个对象 Sender 和 Receiver。像部署图中的描述一样,它们分布在不同的节点上。两者分别拥有一个简单的相关状态图对它们的行为进行描述。像活动图中的描述一样,两个状态图都是并发执行的。Channel 子系统向其所处环境提供 send 和 receive 操作。

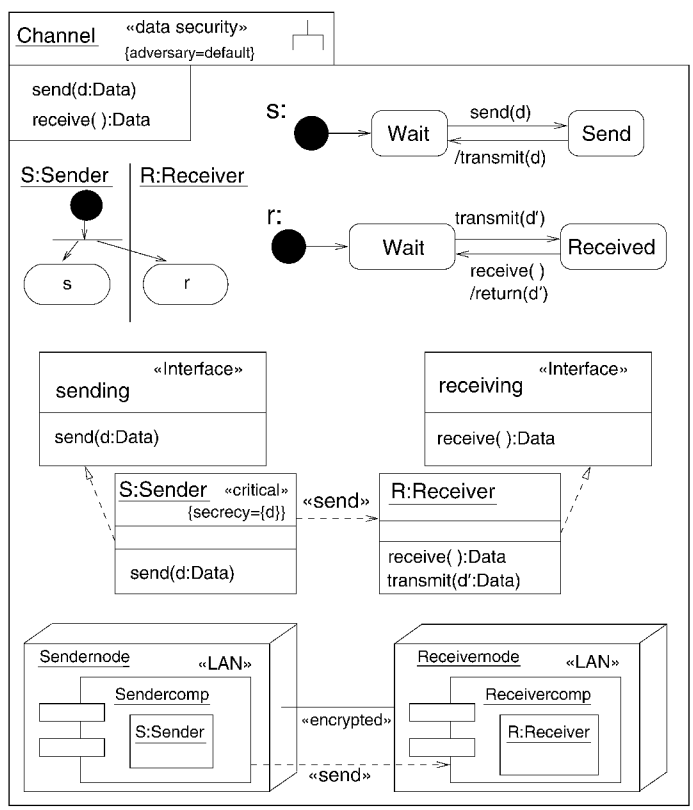

图 3.7 子系统

3.2.8　UML 扩展机制

UML 提供三个主要的轻量级语言扩展机制：构造型、标记值和约束[UML03]。在这里我们不考虑使用元模型扩展的重量级方法。构造型对新的模型元素类型进行定义，方法是对 UML 元模型中现有的类型或类的语义进行扩展。构造型标记包括写在≪ ≫中的构造型名称，附带被扩展的模型元素。然后对模型元素按照该构造型含义进行解释。例如，图 3.7 中附在对象 Sender 上的构造型≪critical≫(将在第 4 章中定义)。从 UML 1.4 版本开始，早期对给模型元素最多只能指派一个构造型的限制被取消掉了[UML03]。

一种显式定义一个属性的方法是给一个模型元素附加一个标记值。标记值是名字-值对，其中的名字记为 tag。对应的符号是{*tag*＝*value*}，其中标记名为 tag，对应的 value 赋值给 tag。标记既可以定义数据值，也可以引用到其他模型元素，前者称为数据标记值，后者称为引用标记值。如果值是一个布尔型，通常省略为{*tag*＝*false*}，并且直接用{*tag*}代替{*tag*＝*true*}。如图 3.7 中标记值实例所示，其中标记{sec *recy*}及其值域{*d*}与附加到 Sender 对象上的构造型≪critical≫关联。

另一个向模型元素添加信息的方法是附加约束 constraints。这些约束必须由图中相关部分实施。如图 3.7 中构造型≪data security≫就有一个与之关联的约束(相关定义见第 4 章)。

构造型附上标记值和约束可以作为被构造模型元素的假属性。之所以被称为假属性值是因为它们的语义不在 UML 定义范围之内。从 UML1.4 开始[UML 03]，所有由这种特定构造型标识的模型元素除了接收标准 UML 中拥有的属性、关联、超类等元素之外，也接收对应的标记值和约束。

为了构建 UML 的扩展，需要将构造型、标记值和约束相关的定义放入同一个模板(profile)中[UML03]，即一个构造型包(在文献[CKM+99]中也称之为 prefaces)。一个模板(profile)：

- 标识 UML 元模型的子集。
- 给这个子集加入“良构型规则”，即一组约束。
- 用自然语言给出已识别子集上规则的语义。
- 列出公共模型元素。

UML 扩展机制实例包括支持软件开发过程的 UML 模板(profile)[UML 03]、支持事务建模的 UML 模板(profile)[UML 03]，以及实时扩展[SR98]和框架扩展[FPR00]。

3.3　UML 模型分析

在 UMLsec 模板(profile)的定义中，人们需要对 UML 模型(对非常敏感的安全需求进行建模)中的约束进行规范。要自动检测这些模型，就要引入一个可分析模型，以表达 UML 片段的行为语义。在本节中，我们对这种用来形式化规范 UMLsec 模板中约束的模型属性进行定义和解释。为了进行安全性分析，那些面向安全性的构造型中的、安全相关的信息被集中在一起。在第 8 章中，我们以第 7 章引入的 UML 机制为基础，给出了本书中使