基于 BIP 框架的 TinyOS 核心组件建模与验证

摘　要　TinyOS(Tiny Micro Threading Operating System) 是一款基于组件体系结构、面向低功耗无线传感网设备的操作系统。TinyOS 的可靠性至关重要，软件形式化方法通过形式化的程序逻辑来验证一个程序是否满足已给定的规范，从而保证程序实现与形式化规范的一致性。对于代码规模相对较小但设计复杂的 TinyOS 而言，形式化验证是保证其可靠性的一种有效方法。BIP(行为—交互—优先级框架，Behavior-Interaction-Priority) 是一种基于组件的通用系统级形式化建模框架，提供了软件开发生命周期中从开发到验证的有效支持。本文在充分分析 TinyOS 核心功能需求的基础上，基于 BIP 形式化框架，提出了对 TinyOS 核心组件进行建模和验证的方法。对 TinyOS 的调度机制、传感器等核心组件进行建模，并搭建了模拟验证环境；进行了常规属性检查和模型参数检查两类形式化验证工作；总结了对此类系统进行分析和建模的一般原则。通过以上建模和验证工作，确保了上述核心组件的可靠性和功能正确性，提高了整个 TinyOS 的系统可靠性。
关键词　BIP；TinyOS；形式化方法；形式化验证

Modeling and Verification of TinyOS Core Components Based on BIP Framework

Abstract　TinyOS (Tiny Micro Threading Operating System) is a component-based architecture operating system for low-power wireless sensor network devices. The software formalization approach verifies that a program satisfies a given specification by formalizing the program logic, thus ensuring that the program implementation is consistent with the formal specification. For TinyOS, which has a relatively small code size but a complex design, formal verification is an effective way to ensure its reliability. BIP (Behavior-Interaction-Priority framework, Behavior-Interaction-Priority) is a component-based generic system-level formal modeling framework that provides effective support from development to verification in the software development lifecycle. In this thesis, based on a full analysis of the core functional requirements of TinyOS and based on the BIP formal framework, we propose a methodology for modeling and verifying the core components of TinyOS. The core components of TinyOS, such as scheduling mechanism and sensors, are modeled and a simulation verification environment is built; two types of formal verification work, regular attribute checking and model parameter checking, are performed; and the general principles for analyzing and modeling such systems are summarized. Through the above modeling and verification work, the safety and functional correctness of the above core components are ensured, and the system reliability of the whole TinyOS is improved
Key words　BIP; TinyOS; formal methods; modeling; verification

1　引言

现代软件系统的规模变得愈发庞大，功能变得愈发复杂，复杂的结构特性为软件缺陷的发现和排除带来困难，依赖输入输出的传统的软件测试对庞大的软件体系进行测试用例覆盖的难度变得越来越大。建模和基于模型检测的形式化方法（Formal Methods）的引入使从根本上保证软件的正确性成为可能。

TinyOS[1] 是一个典型的基于组件概念进行设计和开发的系统，以开源方式向学术界和商业界公布。TinyOS 已经在工业界得到了广泛应用，其可靠性会对这些工业场景下的安全生产产生深刻影响。当前对于 TinyOS 的功能正确性和可靠性的研究较少，开展这方面的工作具有很大现实意义。从理论层面分析，TinyOS 表现出典型的组件化的结构特征，对具有这样的特点的软件系统开展其功能正确性的研究，有利于扩宽对同类特性软件系统可靠性展开研究的途径。在 TinyOS 中一些组件对系统的正常功能运行有着决定性影响地位的，它们被划分核心组件，软件的正常运行极大程度取决于这些组件是否依据功能规范正确运行，对 TinyOS 的分析和研究一般围绕它们展开。

现代软件系统现代软件组件之间产生的交互关系可能带来巨大的软件复杂度，会使软件产生不可预期的行为的概率增大，TinyOS 充分体现了现代大型软件系统的这些特点。目前确保此类复杂的嵌入式系统可靠性的方法从硬件的角度来说主要有采用高可靠器件及冗余容错设计。从系统的角度来说主要有仿真和测试。冗余容错设计基于硬件，本文不作讨论。测试和仿真并不完备，它们只能针对输入和操作条件的一个子集对系统进行评价。此外，系统中有些错误只有当某些特定的事件按一定顺序发生后才会出现，不可重复。使用测试的方法来找这类错误需要尝试大量测试用例，由于不可重复性，这些测试用例的价值很有限。

为了分析这样的庞大系统的功能及其交互的可靠性，基于组件的建模技术（Component Based Modeling Method）得到发展，作为仿真和测试方法的重要补充。BIP[2] 作为一个基于组件的复杂系统建模和验证框架，在验证严格的系统设计中发挥重要的作用。BIP 框架是一个通用的系统级形式化建模框架，支持层次化结构，并包含一套支持建模、模型转换、仿真、验证和代码生成的工具集。BIP 框架已在嵌入式系统中得到了较广泛的应用，包括对实时系统的建模、对同步系统的建模[3]。

本文目的为通过 BIP 框架，对开源嵌入式操作系统 TinyOS 中的代表性核心模块源码功能实现进行建模和分析，开展对 TinyOS 操作系统核心源码的形式化验证工作。本文关注的核心组件包括调度组件、Sensor 和 Timer 等组件，建模和验证的工作也围绕着这些核心组件进行。通过分析其实际需求特征，以求建立描述功能特征的形式化模型，证明待验证的代表性核心模块 TinyOS 系统代码实现确实符合形式化的功能规范，从而保证该操作系统的可靠性。

本文的工作流程如图1所示。首先根据 BIP 的语义，将目标对象抽象成对应的 BIP 结构，然后使用工具进行模型建立和编译，进行模型建立工作；根据得到的 BIP 模型的不同特征，选择适合的建模方法进行验证；最后根据验证结论，对模型正确性进行确认。

本文的主要贡献如下：

(1) 将形式化方法应用到 TinyOS 系统的可靠性保障过程中，使用一套形式化的工具和方法，验证了涉及到的 TinyOS 核心组件是功能正确的。

(2) 给出了使用 BIP 框架对 TinyOS 进行系统级建模的核心流程，针对 TinyOS 系统建立了统一的 BIP 模型。

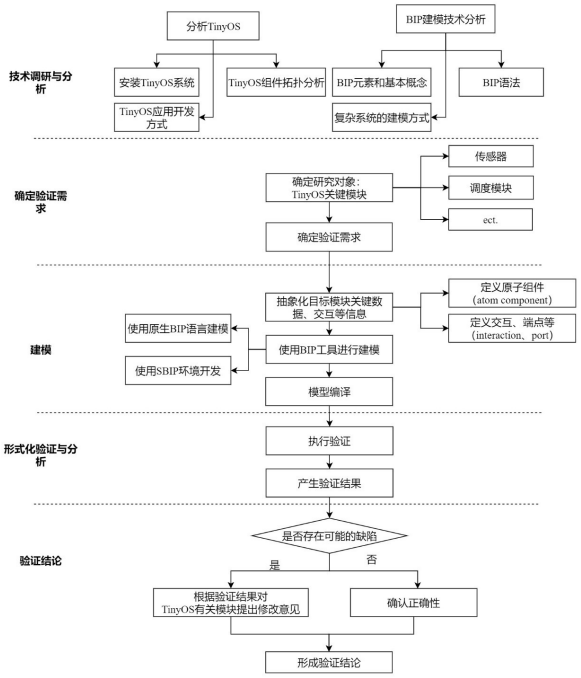

图 1　使用 BIP 对 TinyOS 操作系统建模与验证执行框架

(3) 总结了一套使用 BIP 框架对基于组件的系统建模应遵循的原则及技巧，包括如何对系统进行分解与组合、如何选择恰当的抽象层次、如何描述待验证属性以及如何验证系统是否满足需求。

2　研究背景及相关技术

本节介绍本文涉及到的相关技术方法和所使用的工具软件，包括介绍本课题的研究对象 TinyOS 的体系结构以及抽象体系，和本文所使用的形式化建模工具，即行为-交互-优先级 BIP。

2.1　基于组件的软件形式化建模方法

形式化建模方法已经在不同阶段、以不同形式在现代软件系统的开发与测试过程中得到运用，基于组件的建模技术（Component-Based Modeling Method）不断发展。典型的系统级建模及设计工具有 Berkely 大学的 Ptolemy[4]、荷兰 Eindhoven 大学的 POOSL[5] 和法国 Verimag 实验室的 BIP。其他代表性的形式化工具有结构体系设计语言 AADL[6]、集成模型检查工具 UPPAAL[7]、形式化验证工具 SCADE[8] 等。

使用形式化验证对一些软件系统进行分析的工作已经得到广泛开展。N.Drawel 等人[9] 提出了一个形式化的框架，允许个人和组的代理人推理他

们对其他代理人的信任并提出了一种分支时态逻辑 BT，包含了表达诸如每个人信任、分布式信任和传播信任等概念的算子。W.Nam 等人[10] 提出了一种新的形式验证技术来分析区块链智能契约并使用 ATL 模型检查，将用户和智能契约之间的交互表示为一个双人游戏，并使用 MCMAS 验证想要检查的属性。在物联网领域[11] 和车辆自动驾驶[12] 方面，形式化验证已经得到的实际应用并证明了其价值。BIP 框架以其强大的组件化特性也不断成为该领域的重要研究对象，Wan 等[13] 人介绍了如何使用 BIP 框架对一个星载软件中的数据处理单元（DPU）进行形式化建模。Tang 等人[14] 介绍了利用 BIP 框架对一个民用飞机子系统进行故障建模的方法。

2.2 TinyOS

TinyOS 通过组件的形式、使用 nesC 语言进行应用的构建和开发，允许开发者可以根据不同的场景进行组件的选用。TinyOS 的大小以精简著称，其核心代码仅为 400 字节左右，很好地适应了节点存储资源受限的情况。

一个典型的 TinyOS 应用程序由一个或多个组件构成。在 nesC 语言中，组件除了一类用于实现具体的程序功能、实现接口声明的能力并对外提供接口的模块组件（module）外，还有一类用来声明功能实现上有关联的组件之间的关系、指引 nesC 编译器在编译时进行根据这些声明连接的配置组件（configuration）。TinyOS 操作系统本身及其应用程序都由组件构成（或称为“导通”，wiring）。图2描述了一个节点内运行的 TinyOS 结构。模块组件的结构如图2左侧所示，通常包括一组命令函数（Task Handler）、一组事件函数（Event Handler）、若干任务函数（Command Handler）、若干普通函数和一个描述状态信息其数据结构的框架；图2右侧表示 TinyOS 提供了任务和事件两级调度机制以及以无线通信、时钟、传感器为代表的硬件机制。

2.3 BIP

BIP 是一种基于组件的编程语言，用于复杂系统的建模和编程。在 BIP 中，一个系统表示为如图3所示的三层结构：

（1）由一组行为（Behavior）所定义的组件；

（2）一组交互（Interaction）定义了组件之间可能存在的同步和通信；

（3）一组优先级（Priority），用于解决交互之间的冲突或定义交互时间策略。

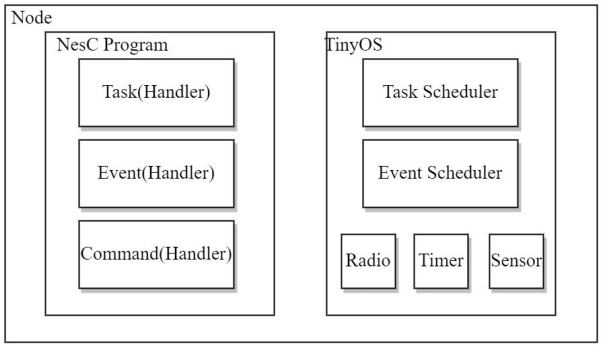

图 2 TinyOS 的结构示意图

BIP 框架从行为、交互、优先级三个方面描述一个复杂系统，下面分别具体阐述三者的概念。

首先是行为。这一层由一个或多个数量的异构组件组成，每个组件都是独立定义的。这些组件在 BIP 的语境下被称为原子组件（或简称原子，atom component）。每个组件可以被模拟成一个有限状态机（FSM），每个原子的内部操作被模拟成一个 Petri 网。一个原子的当前状态由当前所处的状态集合和原子组件的内部变量值表示。原子组件内不同状态（state）之间的转换会更新原子的变量和所处状态的集合。

其次是交互。组件之间的交互关系是通过连接器（connector）连接起来的端口（port）来定义的连接器包含了交互和端口的集合。在 BIP 中，有两种类型的交互得到了定义：广播（broadcast）和强同步（rendezvous），前者表示指定的交互必须包含完备（充当“触发器”的角色）和不完备（充当“接收器”的角色）两种类型的端口，后者指示了所有端口都必须参与交互，而前者根据具体触发和接收的条件不同实现端口的部分参与。一个交互由四个元素描述：参与交互的端口的集合、交互的使能条件（guard）、上行动作和下行动作。

最后是优先级。优先级是为了消除非确定性和执行调度策略，作为在多个可选的的执行路径之间进行选择的手段。当有多个同时启用的交互时，不确定性就会产生，每个交互都可能会导致产生不同的整体系统状态。通常情况下，如果有一个以上的连接器有启用的交互，不能保证哪个交互会执行，我们可以通过组件和端口两个维度来指定交互或端口的优先级。

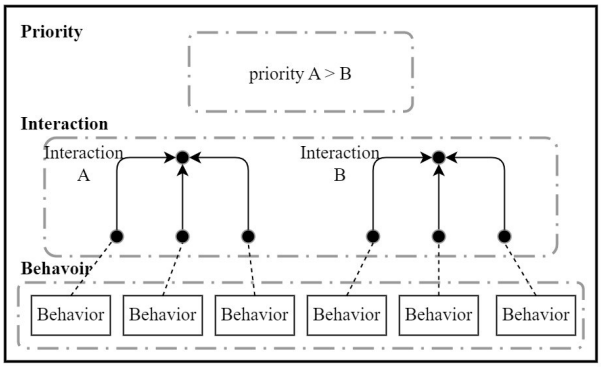

图 3　BIP 的三级结构：Behavior、Interaction 和 Priority

3　基于 BIP 建模的基本方法

建模过程需要关注的核心问题是保证所建立的模型必须与目标系统保持功能维度上的一致。形式化建模工作是在关注特定需求的前提下，对原本的系统进行合理的裁剪和抽象。如图4所示，建模过程采用的是一套可以概括为以自顶向下分解（top-down decompose）和自底向上组合（down-up modeling）为代表的工作方式。

第一步，基于原始研究对象的分解。首先要求对目标系统进行深入的研究。在充分把握住其关键交互、状态迁移和时序等信息的前提下，先根据目标系统的结构和功能来选取某些具有研究价值的维度，根据这些维度对目标系统进行垂直向下的分解。每一次的分解操作都会将研究对象分割成若干个平行的子模块，它们可以完成一定的功能。接下来对这些每个子模块都依照相同的原则递归地继续进行若干次分解操作，直至被分解为更细粒度的、可以独立完成一定功能的子模块。这些子模块某种程度上具有原子性（atomicity）。这些细粒度的子模块在承担独立功能的同时也可以被更上层的一个或多个模块所复用，它们对应着 BIP 中的原子组件概念。原子组件的转化模式示意图见图5。确定最小粒度以停止分解的标准则需要依据实际情况决定，粒度过大过小都不利于建模工作的进行。

第二步，基于原子组件的组合。得到了一定数量的原子组件后，根据目标系统模块之间如调用、中断、通信等交互关系，对这些独立的原子进行自下而上的逐层的组合和连接。为了正确地模拟复杂交互关系，有时候还需要定义连接器端口或组件的优先级。每层的连接对应着原来子模块的一个层级的实现，如此往复即可构建出整体上的目标模型。这些组合和连接关系被表达成 BIP 中的复合组件（compound component），它是由若干个原子组件实例、连接器实例和优先级定义的。一个 BIP 模型需要指定一个复合组件类型作为整个模型文件的入口，通过集成原子类型和连接器的实例组合模型，实现了 BIP 模型的集成。

4　TinyOS 核心组件的形式化建模

本节主要对涉及到的形式化建模过程进行介绍，分别具体地对 TinyOS 中的调度组件、执行机制、计时器和传感器机制进行分析和形式化建模，最后通过一个由上述组件和机制所构成的示例应用串联起所有的 BIP 模型，提供了一个整体的概念。

4.1　TinyOS 调度组件的形式化建模

在一个操作系统中，其调度模块的重要性居首要地位。操作系统能够正确地对其中的操作进行合理的调度，是一个操作系统能够正常运作的基础。对 TinyOS 的形式化工作首先围绕其调度组件展开。

4.1.1　调度机制和调度组件分析

TinyOS 提供了任务和事件两级调度模型。调度机制的示意图见图6。

TinyOS 首先提供了任务调度机制，一些可以承受较大延迟的计算或者操作通常采用这种机制。任务具有：1) 原子性。要么某任务被完整执行完毕，要么不被执行；2) 相互平等。TinyOS 默认的实现没有对任务赋予有实际意义的优先级；3) 非抢占。任务之间不可相互抢占。在 TinyOS-2.x 中，FCFS（First Come First Service，先来先服务）方式被选为默认的任务调度算法，即所有提交的任务都必须按照进入任务队列的先后依次进行调度执行。

事件是一种响应硬件中断或软件事件驱动的一种处理机制。当一个硬件模块产生中断请求，此时相应代码中会触发 signal 关键字所标记的一个事件函数的执行，在此过程中，所有与发出信号的事件的相关联的操作都会依据由下到上进行触发和运行。不同于任务的调度，事件的调度是抢占式的。当一个事件触发时，它会抢占所有正在运行的任务、事件或命令（command）执行体，并将被抢占的操作入栈，待事件调度完毕后再恢复被抢占的操作的调度执行。

4.1.2　Task 调度组件建模

对 TinyOS 任务调度组件的建模以 SchedulerBasicP 中的代码为基础，并结合 TinyOS 的运行实际进行。根据 TinyOS 的装配组件 TinySchedulerC

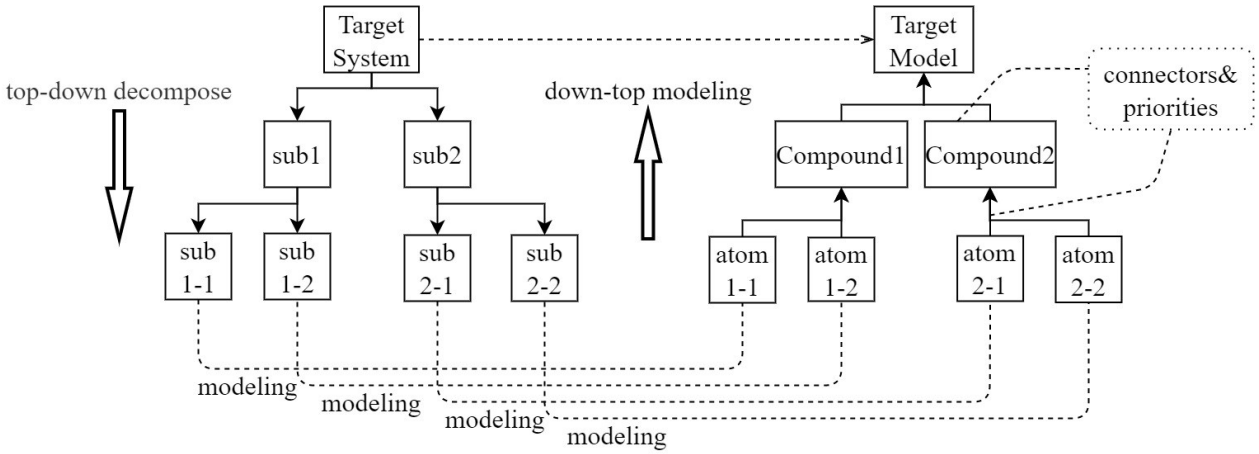

图 4　建模过程中的自顶向下分解和自底向上组合方法

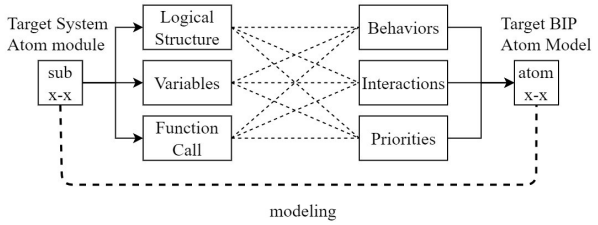

图 5　BIP 原子组件的转化模式

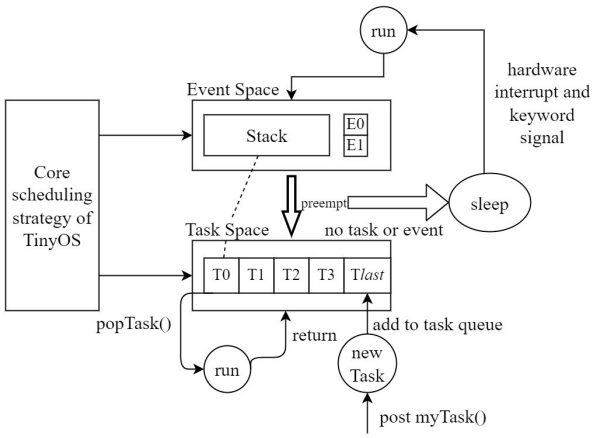

图 6　TinyOS 调度原理

中的定义，核心的调度接口 Scheduler 的功能实现是由模块 SchedulerBasicP 提供。根据自顶向下分解的原则，可以将该模块对应为 BIP 模型中的一个原子组件

BIP 模型示意图见图7，这是一个有限状态机模型，圆内的是所处的状态，箭头标识着状态之间发生转移动作、守卫条件等。对于任务调度机制 TaskScheduler，它的状态可以抽象为三个：BOOT、FREE 和 BUSY。下面将分别阐述每个状态代表的实际意义：

FREE 状态表示任务调度器处于空闲状态，即当前的任务队列为空，没有新的任务进行提交。实现上任务调度器有一个无限循环，不断地从队列中取出任务。无任务时，整个操作系统会进入低功耗状态，以适应无线传感网的低功耗性能要求。

BUSY 状态表示正在进行任务调度，即任务队列中有任务存在，或者当前状态下有正在被执行的任务。FREE 和 BUSY 两个状态只是用于代表当前是否有任务正在被运行，在任务队列未满的前提下，不论哪个状态下都不影响持续接受新任务的提交。

下面介绍模型中定义的端口，以及它们标记的状态转移时所要进行的动作。

init 端口标记了从 BOOT 状态迁移到 FREE 状态的动作，对应的是 TinyOS 操作系统内核启动流程中 Scheduler.init() 接口实现的功能。该端口应该与操作系统的启动序列进行同步绑定。这一步需要初始化任务调度器所需的数据结构，如 FIFO 队列、寄存器等。

post 端口反映了外部的任务执行体向任务调度器提交注册并任务，任务加入调度序列的操作。它可以在 BUSY 状态或 FREE 状态下触发一个自身同状态的迁移，这意味着不论调度器处于空闲或者是忙碌时，都可以接受新任务的加入（前提是任务队列还可以接受新的任务提交），放在任务队列中等待调度。

beg 端口和 fin 端口分别表示任务处理的开始和结束控制。当任务队列不为空时从任务队列中根据 FIFO 原则取出一个任务开始调度，状态也从空闲迁移到忙碌。当任务处理结束时，根据其发出的信号，任务调度器结束调度，调度器恢复空闲状态，等待新的任务的开始。

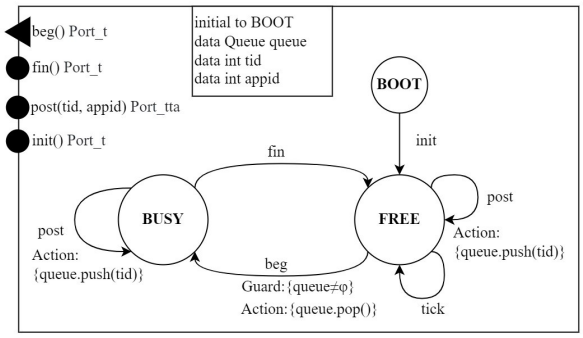

图 7　BIP 模型：Task 调度器

4.2　TinyOS 计时器和传感器机制的形式化建模

在 TinyOS 中，计时器 Timer 和传感器 Sensor 是用户感知比较明显而且比较常用的硬件相关的机制，TinyOS 也为用户提供了易用的组件。

4.2.1　计时器机制的分析和建模

在 TinyOS 的源码中，计时器相关的代码支持和实现在目录/tos/lib/timer 下。计时器的三个基本属性是精度（如毫秒、微秒等）、宽度（32 位、64 位等）和准确度（与其声称提供的精度的一致性程度）。

TinyOS 没有试图使用独立于平台的方式来捕捉各种定时器硬件的多样性。与之相反的是，每个平台都应该通过 HPL 的组件及其提供的接口，以及适当抽象程度的 HAL 级别来发布这些功能。本文更多关注该操作系统提供的通用的功能实现和普遍流程，故并不会对硬件平台的具体实现机制做深入探讨，聚焦在 HIL 层进行分析。

Timer 组件的加入使得 BIP 模型需要适应时间层面约束。基本的 BIP 编译器适配的模型不能很好地体现时间约束这一特点。为了解决此问题，我们在 BIP 模型的扩展中选用了带时效性的 BIP 模型（real-timed BIP）[15]。流程图8展示了实时 BIP 引擎的工作方式。概括地说就是当在原 BIP 引擎的基础上加上时间的约束，当一个原子内的动作满足基本约束要求和时间要求时，进行对应的状态变迁和交互。

我们把计时器机制的对应 BIP 原子组件声明为 Timer。BIP 结构图如图9所示。在这个原子组件里，有三个状态被声明，分别为 INIT、EXE 和 DONE。INIT 状态表示计时器的初始化操作已经完成，将 clock 变量置为 0，设置时间单位等。EXE 状态表示计时器正在正常工作中，DONE 状态计时器已经被终止。

Timer 原子组件向外提供 4 个端口。post 端口

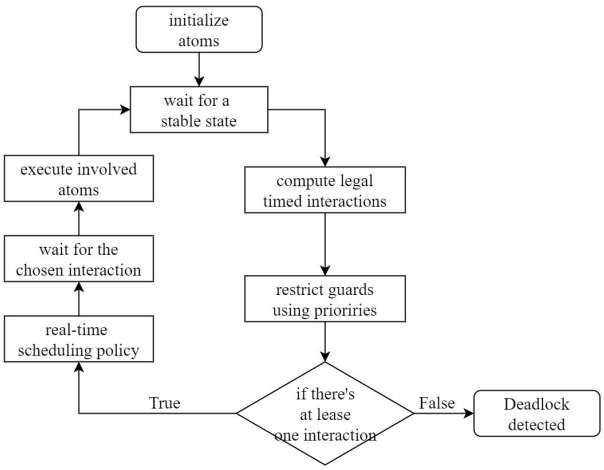

图 8　RT-BIP 引擎的工作流程

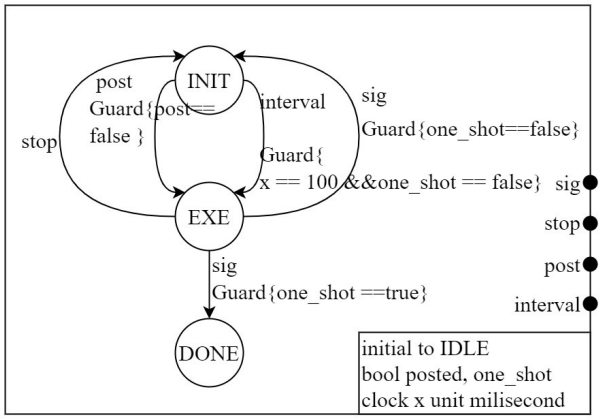

图 9　RT-BIP 引擎的工作流程

用于接受新的计时任务的提交，触发该端口后将使模型从 INIT 状态迁移到 EXE。sig 端口表示时间到达触发事件回调。根据计时器是否是一次性的计时器分为两种触发路径，若条件 is_one_shot 为真，模型状态从 EXE 迁移到 DONE，计时结束；为假则重新回到 INIT 状态开始下一次计时。interval 端口是特为周期性计时器准备的，其限制条件里加入了 clock 变量，只有当该变量等于设置好的周期长度才会触发。stop 端口用于结束整个计时器。注意完成一次操作后要及时把 clock 变量置为 0，重新开始新的计时周期。

4.2.2　传感器机制的分析和建模

传感器的原理是通过各种机制，将外界的信息（如温度、湿度等）模拟输入信号转换为离散数字输出信号的设备，通常将电压转换为二进制数。尽管传感器的类型多种多样、提供的数据接口和返回数据结构各不相同，TinyOS 为用户提供了统一的数据采集接口来使用这些传感器并得到其返回的数据。在 HIL 层中提供了一系列的数据采集接口，

例如 Read、ReadStream 和 Get，传感器根据其采集能力提供这些接口，这些接口通过灵活的导通可以对代表各种类型的传感器组成。然而统一的接口模式会牺牲一些传感器硬件各异的特性，如果需要高频或非常精确的采样，它必须使用 HAL 提供的能力。本文中我们只对通用的模式进行研究，暂不涉及 HAL 层。

为了适应 TinyOS 的分相机制，对于传感器机制进行形式化建模时，会产生如下结构组成：传感器机制对应的 BIP 原子组件名为 Sensor，共有两个状态：E1 和 E2。前者作为初始状态表示未读取数据，后者表示已经通过 ADC 机制读取到数据并交由调用者处理。提供了 sig、read 和 test 三个端口——read 端口代表从外界获取数据的交互；sig 表示向调用者提交数据，test 端口用于向外界提供传感器当前是否可用的状态，同时产生自旋，防止在此产生死锁。为了使我们的传感器组件能够模拟真实运行状态，我们还提供了一个模拟环境产生数据的原子组件 Environment，它能够产生随机数作为 Sensor 的输入数值，它们通过连接器 Conn_Sen_Ch_Beg 进行关联，并进行数据交换。示意图见图10。

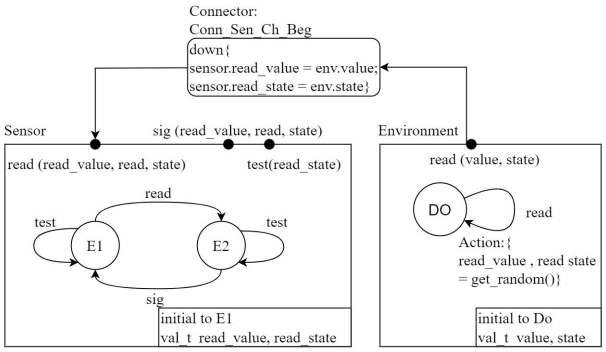

图 10　BIP 原子组件：Sensor、Environment 及其连接器

4.3　核心组件整体建模

经过对 TinyOS 核心组件的 BIP 建模后，我们得到了若干个孤立的 BIP 原子组件。下一步将实际存在的组件之间的调用和关联关系转化为 BIP 模型中对应的连接，从而把这些孤立的 BIP 原子组件组合成一个完整的 BIP 复合组件，这样的一个复合组件即为原示例应用的 BIP 模型。

4.3.1　示例应用 BasicMAC

为了使前面几节建模得到的 TinyOS 调度、时钟和传感器等组件和机制都能够充分发挥作用，示例应用必须能够有效地体现上述几个组件的特点。我们选取 TinyOS 的官方示例程序库中的一个名为 BasicMAC 的应用作为目标应用。该应用的功能是对两个节点进行点对点通信的情况下，要求每个节点每隔 1s 将从光照传感器读到的测量值，通过无线通信发给另外的一个节点；接收到发送方发来的数据包的接收方根据传递的数值来点亮设备上的 LED 灯。示意图见图11。

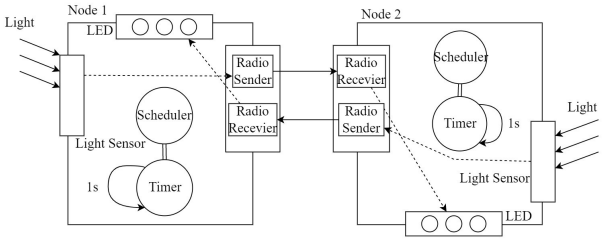

图 11　示例应用 BasicMAC 示意图

该应用的用户组件由配置 BasicMAC 和模块 BasicMACM 组成，前者实现各组件之间的导通关系，后者用于实现应用的功能。配置 BasicMAC 中计时器和光照传感器分别由 TimerMilliC 和 PhotoSensorC 组件提供，在模块 BasicMACM 中它们分别与 Timer 和 Read 接口导通，这样为用户的使用提供了统一的操作方式。ActiveMessageC 组件声明了无线传输光照值的能力，且通过 AMSender 和 AMReceiver 组件相导通的 DataMsg 和 ReceMsg 接口控制发送和接收。

4.3.2　BasicMACM 的 BIP 建模

BIP 中的复合组件由一组原子或其他复合组件的实例、一组连接器的实例和若干优先级定义构成，代表着根据一定规则组合若干组件功能的实体。定义一个复合组件 BasicMACM 作为一个完成该应用程序功能节点的抽象。根据应用程序所需要的功能，该复合组件所用到的原子组件包括：启动流程组件 NodeBoot、任务调度器 TaskScheduler、事件调度器 EventScheduler、计时器 Timer、传感器 Sensor，环境的抽象 Environment 和 LED 组件 Led、发送方组件 RadioSender 和接收方组件 RadioReceiver。声明这些实例的语句形式为“component [ComponentType] [InstanceName](…args)”。上述原子组件的原理和定义均在本章前面章节进行了介绍。除此以外，我们还要建立一定数量的命令、任务和事件的执行体 Handler 原子组件来承载实际的功能，它们也是整个模型的主体。

完成组件的实例化后，下一步要定义连接器。连接器是无状态实体，通过其接口端口实现一组组件之间的交互，声明连接器的实例语句形式为

“connector [ConnectorType] [InstanceName]”。在复合组件 BasicMACM 中，共定义了 27 个连接器实例，来连接各个组件的众多端口。

整体模型中部分连接器关系见图12，图中方框表示一个原子组件，黑色圆点表示原子组件上的端口，端口之间的连线为定义在端口上的连接器实例。这样就定义了整个系统的行为模式。

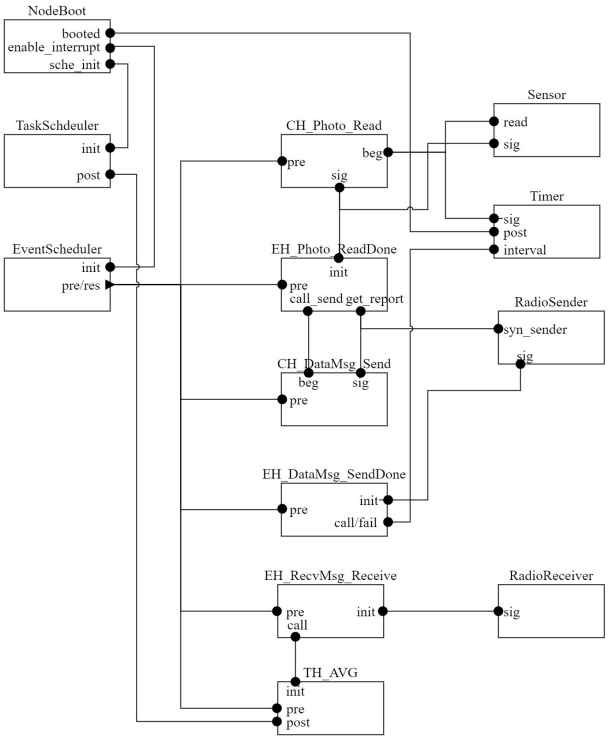

图 12　BasicMAC 整体模型中部分连接器关系

整个模型 BIP 代码共计超 1800 行，其中端口类型 9 个，连接器定义 14 个，原子类型 16 个，复合组件 2 个。C/C++ 辅助文件共 6 个，提供包括队列和栈的数据类型支持，提供了 17 个辅助函数。

5　TinyOS 关键组件的形式化验证与评估

本节将对上节产生的 TinyOS 由核心组件构建出的一个典型应用的 BIP 模型进行形式化验证。针对目标系统和 BIP 模型的特点，我们的形式化验证分为常规属性检查和模型参数检查两部分。在此过程中将会介绍这两种形式化验证的方法和过程，最后根据实验结果给出形式化验证的结论

5.1　常规属性检查——以死锁检查为例

一般来说，模型的正确性可以由一些常规属性进行概括，包括无死锁、互斥、顺序性和原子性等，这些常规属性的检查都遵循着类似的验证流程。在本课题中，对于这部分常规属性的检查，我们主要关注 BIP 模型的无死锁特性。我们将以死锁检查为例，介绍这部分工作的流程，并给出检查结论[16]。

死锁（Dead Lock）是操作系统中的一个常见的概念。通常的定义为两个或两个以上的实体集合中的每一个，在其执行过程中，出于某些原因无法得到运行所需的资源，从而导致整个系统都无法继续运行下去的一种错误。在 BIP 的语境下，死锁指的是 BIP 模型在执行状态迁移时，无法从待选状态的集合中选出下一个可用的交互的状态（这样的集合很可能是空集）。死锁检查是对整个模型进行统计学上的模拟和仿真，验证在指定条件下 BIP 模型不会出现死锁，陷入死锁状态。

在进行死锁检查时，BIP 引擎提供的随机能力在执行次数和步数足够多时可以覆盖可能出现的状态组合，从而完全确定系统有无陷入死锁状态。一个正确的 BIP 模型在运行仿真过程时，在每个状态下都会确保至少有一个可用的交互或端口是可用的。当在前一个状态下因为不满足守卫条件或者原子组件的有限状态机状态未就绪导致没有可选的状态可供迁移时，BIP 模型会认为这条执行路径陷入了死锁，这样的模型没有通过死锁检查。

经过 BIP 引擎的编译得到了大小为 8.4MB 的可执行文件 system。如表1所示，我们对 system 模型进行了 10^2、10^3、10^4 和 10^5 四个数量级的仿真死锁检测。经历了不同步数的验证后，实验结果显示 BIP 运行的结果均为“stop(reach the limit of given states)”，模型均运行到我们指定的步数限制。由此，我们可以断定，在 10^5 数量级以下，TinyOS 的核心组件 BIP 模型是死锁安全的，原系统的基本运行机制是正确的。考虑到该量级的大小，加上 BIP 引擎的随机性，足以覆盖绝大部分可能出现的路径选择。据此，有理由推断，在一般情况下，TinyOS 的核心组件是死锁安全和功能正确的。

5.2　模型参数检查

在进行形式化验证时，我们往往不仅会关注以死锁检查为代表的常规属性检查，来验证整体的运行模式的正确性，我们还会关注模型局部是否符合预期。整体的正确只在宏观上维度上保证了系统的正常运行，但是对于一些微观上的细节，可能会有其他方面的顾虑。比如表达某个状态的变量值超出了预期值，但所超出的程度不影响整体的正常运行。为了排除这种潜在的风险，我们需要对一些关

表 1　死锁检测仿真结果

序号	仿真步数（周期数）	耗时	BIP 引擎运行结果	是否出现死锁
1	100	600ms99ns	stop(reach the limit of 100 states)	否
2	1,000	6s900ms334ns	stop(reach the limit of 1,000 states)	否
3	10,000	1min9s900ms2μs106ns	stop(reach the limit of 10,000 states)	否
4	100,000	62min32s36μs16ns	stop(reach the limit of 100,000 states)	否

键的模型参数进行统计性质的检查，以确保微观上模型也是正确的。

在我们之前进行的 BIP 形式化建模工作中，每个原子组件和复合组件内都定义了大量的变量来对模型运行时的状态进行保存。我们可以通过这些输出的信息来确定我们关注的变量状态的值。为了能够对大量的运行轨迹中的变量进行统计意义上的观察和处理，BIP 为我们提供了模型参数检查功能。

要使用模型参数检查功能，首先要对关键参数的功能性需求进行形式化转化，也就是说使用一套特定的表达方式将这些约束表达出来。BIP 中允许我们使用线性时态逻辑式（Linear Temporal Logic, LTL）来进行形式化描述。LTL 是一种能表达时间概念的特殊时态逻辑，常用来精确表示模型的动态语义。模式如图13所示

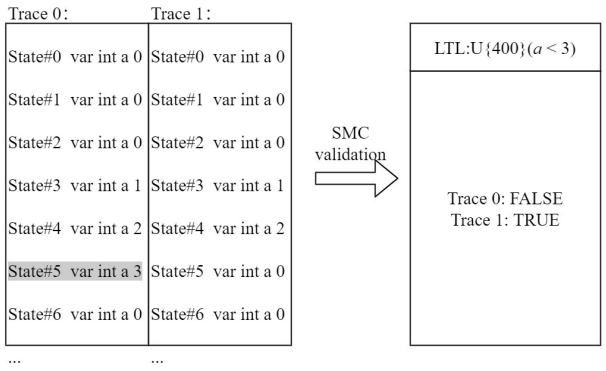

图 13　LTL 规约与参数检查模式

针对调度机制，4.1节中我们介绍了 TaskScheduler 原子组件。我们在设计编码时设置了一个布尔标志位 FALG_IS_IDLE，如果有新任务或新的事件到来时，BIP 引擎过多地选择了进行从 IDLE 状态到 IDLE 状态的自旋，没有针对新的事务进行及时处理导致系统一直处于空转状态，该标志位会置为真。这解释了为什么死锁验证通过的情况下还有可能出现系统正确性问题。同样地，对于 Sensor 和 Timer，我们都设置了类似的标志位以监测他们是否出现运行时异常，详细的形式化规约见表2

以 1 号需求为例，要求检验在 1000 个周期中，最终都没有出现空转导致无效运动的情况。我们使用 BIP 提供的默认配置进行检测。结果如图 5-7 所示：对于 1 号需求，共执行了 9 次，得到 9 个轨迹，每次包含 1000 个周期的仿真。以上仿真的结果均为真，满足功能规约。

所有功能需求的验证结果如表3所示。经过实验，所有定义的规约在 1000 个周期的条件下均为真。也就是说，这些错误的场景在上述限定条件下不会出现。

经过模型参数检查，模型的执行细节得到了一定保证。可以推断，TinyOS 的核心组件是功能正确的。

6　总结与未来工作

TinyOS 作为一种被广泛运用于实际产生中的嵌入式操作系统，其功能可靠性受到格外的关注。形式化建模和验证方法能够对研究对象进行关键机制和性质的转化和提炼，通过对基于原系统机制的模型的行为特征开展研究，可以清晰而有效地验证原系统是否满足一定的功能规约，从而判断它是否是可靠的。BIP 框架作为一种基于组件的形式化建模与验证框架，它通过行为、交互和优先级三个维度描述一个系统的行为特性，具有描述能力强和自动化程度高的特点。

本文基于 BIP 形式化建模和验证框架，提出了一种对 TinyOS 核心组件进行建模和分析的方法。整个过程分为建模和验证两步。

（1）BIP 建模。首先我们提出了一种针对复杂软件系统分析和使用形式化建模手段将原系统转化为相应模型的基本原则和方法，概括为“自顶向下分解”和“自底向上建模”两个部分。基于这种方法，我们针对 TinyOS 的代表性核心组件和机制进行了分析和建模，包括其任务调度组件、计时器和传感器四大部分。对于其中的每一个核心组件，

表 2 详细形式化规约

序号	来源 BIP 组件	描述	LTL 表达式
1	TaskScheduler	是否一直处于空转状态	F{1000}(!FLAG_TS_IS_IDLE)
2	Timer	初始化失败	F{1000}(!FLAG_TIM_FAILED)
3	Sensor	读取值失败	F{1000}(!FLAG_SEN_FAILED)

表 3 形式化需求的验证结果

序号	来源 BIP 组件	LTL 表达式	验证结果
1	TaskScheduler	F{1000}(!FLAG_TS_IS_IDLE)	真
2	Timer	F{1000}(!FLAG_TIM_FAILED)	真
3	Sensor	F{1000}(!FLAG_SEN_FAILED)	真

首先我们对其展开了原理的分析，基于分析提炼出基本的行为模式转化为有限状态机模型，然后生成了对应的 BIP 原子组件。最后我们通过一个典型应用的示例，将上述所有组件综合运用，形成了一个可被验证这些组件正确性的 BIP 复合模型。以上过程中，我们详细介绍了如何使用过 BIP 进行模型开发的过程。

（2）BIP 模型的验证。针对得到的 BIP 模型的特性，我们进行了两个维度的形式化验证工作。首先是对于常规属性检查，以死锁检查为代表介绍了其原理和操作过程，最后对我们的模型进行了验证实验，得到了结论。其次，对于具有重要作用的模型属性，我们介绍了模型参数检查的意义和具体方法，介绍了形式化规约表述语言 LTL，并选取了三个重要的功能规约进行参数检查，最后得出实验结果。

两个形式化验证结果表明，我们的模型从整体上表现出了无死锁特性，这保证了原系统组件逻辑的成立；从局部来看，运行时具体的参数都符合功能规约的要求，这体现了原系统的功能属性。我们可以做出判断——待验证的代表性核心模块 TinyOS 系统代码实现确实符合形式化的功能规范，该操作系统的可靠性和正确性得到验证。

未来研究方向可以分为以下几点：

（1）继续精化建模过程。当前的建模过程是依靠人工进行拆解和分析，这种工作这一定程度受限于验证人员的经验和对研究对象的把握程度。未来期待引入自动化建模工具，可以直接将源码进行形式化工作，减少人工因素的干扰，增强模型的可信度。

（2）安全需求描述可以引入更多现实背景。当前我们更多是对该系统的基本运行机制进行分析，未来希望能够针对特定生产场景，根据特定的安全需求，使用更加场景化的安全需求建模方式来提升这项工作的现实性，以期体现更大的现实价值。

（3）机器学习、统计方法与形式化验证的深度结合。SBIP 还提供了大量的基于统计学的模型验证工具。未来可以将机器学习与形式化工作进行深入结合，发挥机器学习统计学上的优势，进一步把模型验证的效率进行提升。

参考文献

[1] LEVIS P, MADDEN S, POLASTRE J, et al. Tinyos: An operating system for sensor networks[J]. Ambient intelligence, 2005:115-148.

[2] BASU A, BENSALEM B, BOZGA M, et al. Rigorous component-based system design using the bip framework[J]. IEEE software, 2011, 28(3): 41-48.

[3] SIFAKIS J. Component-based construction of real-time systems in bip. [C]//CAV. [S.l.: s.n.], 2009: 33-34.

[4] BUCK J, HA S, LEE E A, et al. Ptolemy: A framework for simulating and prototyping heterogeneous systems[M]//Readings in hardware/software co-design. [S.l.: s.n.], 2001: 527-543.

[5] VOETEN J P M. Poosl: An object-oriented specification language for the analysis and design of hardware/software systems[M]. [S.l.]: Eindhoven University of Technology, Faculty of Electrical Engineering, 1995.

[6] FEILER P H, GLUCH D P, HUDAK J J. The architecture analysis & design language (aadl): An introduction[R]. [S.l.]: Carnegie-Mellon Univ Pittsburgh PA Software Engineering Inst, 2006.

[7] BEHRMANN G, DAVID A, LARSEN K G. A tutorial on uppaal[J]. Formal Methods for the Design of Real-Time Systems: International School on Formal Methods for the Design of Computer, Communication, and Software Systems, Bertinora, Italy, September 13-18, 2004,

Revised Lectures, 2004:200-236.

[8] BERRY G. Scade: Synchronous design and validation of embedded control software[C]//Next Generation Design and Verification Methodologies for Distributed Embedded Control Systems: Proceedings of the GM R&D Workshop, Bangalore, India, January 2007. [S.l.]: Springer, 2007: 19-33.

[9] DRAWEL N, BENTAHAR J, LAAREJ A, et al. Formal verification of group and propagated trust in multi-agent systems[J]. Autonomous Agents and Multi-Agent Systems, 2022, 36(1):19.

[10] NAM W, KIL H. Formal verification of blockchain smart contracts via atl model checking[J]. IEEE Access, 2022, 10:8151-8162.

[11] HOFER-SCHMITZ K, STOJANOVIĆ B. Towards formal verification of iot protocols: A review[J]. Computer Networks, 2020, 174:107233.

[12] ALVES G V, DENNIS L, FISHER M. A double-level model checking approach for an agent-based autonomous vehicle and road junction regulations[J]. Journal of Sensor and Actuator Networks, 2021, 10(3): 41.

[13] HAI W, CHONGDI H, YUHUI W, et al. Modeling and validation of a data process unit control for space applications[C]//Embedded Real-Time Software and Systems (ERTS^ 2 2012). [S.l.: s.n.], 2012: 6B-1.

[14] 唐旭东. 基于 bip 的航空控制系统行为故障建模与形式化验证方法[M/OL]. 华东师范大学, 2022. DOI: 10.27149/d.cnki.ghdsu.2022.000005.

[15] SOCCI D, POPLAVKO P, BENSALEM S, et al. Modeling mixed-critical systems in real-time bip[C]//1st workshop on Real-Time Mixed Criticality Systems. [S.l.: s.n.], 2013.

[16] SU C, ZHOU M, YIN L, et al. Modeling and verification of component-based systems with data passing using bip[C]//2013 18th International Conference on Engineering of Complex Computer Systems. [S.l.]: IEEE, 2013: 4-13.

[17] BASU A, MOUNIER L, POULHIES M, et al. Using bip for modeling and verification of networked systems–a case study on tinyos-based networks[C]//Sixth IEEE International Symposium on Network Computing and Applications (NCA 2007). [S.l.]: IEEE, 2007: 257-260.