

团 体 标 准

T/CSNAME 183—2026

绿色智能船舶协同研发平台模型封装技术 要求

Technical requirements for model packaging of green intelligent ship collaborative
R&D platform

2026 - 10 - 23 发布

2027 - 01 - 23 实施

中国造船工程学会 发 布

目 次

前言 1

1 范围 1

2 规范性引用文件 1

3 术语和定义 1

4 缩略语 2

5 GISSMU 生成的通用要求 2

5.1 核心元数据 2

5.2 文件结构与编码 3

5.3 接口与线程安全 4

5.4 二进制与兼容性 5

5.5 扩展与调试 5

6 GISSMU 生成要求 6

6.1 GISSMU 代码 6

6.2 变量 7

6.3 GISSMU 结构和文件说明 7

6.4 接口要求 8

6.5 GISSMU 分布式部署 9

7 验证和测试要求 9

7.1 GISSMU 验证规则 9

附录 A（资料性） 术语和定义补充内容 12

A.1 GISSMU 文件内容 12

A.2 GISSMU 文件生成流程 12

A.3 协同仿真计算流程 13

A.4 变量类型 13

A.5 变量属性 14

A.6 模型描述文件元素 14

A.7 模型变量属性 14

A.8 接口函数 15

A.9 接口属性 15

附录 B（资料性） 船舶动力定位系统示例 16

B.1 船舶动力定位系统概述 16

B.2 生成 GISSMU 16

附录 C（资料性） 验证示例代码 21

C.1 填充本体示例 21

C.2 分类验证示例代码 24

前 言

本文件按照GB/T 1.1—2020《标准化工作导则 第1部分：标准化文件的结构和起草规则》的规定起草。

请注意本文件的某些内容可能涉及专利。本文件的发布机构不承担识别专利的责任。

本文件由中国造船工程学会船舶标准化专业委员会提出。

本文件由中国造船工程学会归口。

本文件起草单位：中国船级社、水上载运装备安全研究院、上海船舶运输科学研究所、大连海事大学。

本文件主要起草人：孙旭、王剑凡、李忠建、乔继潘、神和龙。



绿色智能船舶协同研发平台模型封装技术要求

1 范围

本文件规定了绿色智能船舶协同研发平台中GISSMU 模型的核心元数据、模型描述文件、接口调用时序、打包与代码生成加载、参数和变量处理等模型封装技术要求。

本文件适用于绿色智能船舶协同研发平台下，基于 FMI3 标准的船舶仿真模型（GISSMU）封装、打包、交付与集成调用；可用于船舶多学科、多专业仿真模型的协同接入与联合仿真。

2 规范性引用文件

下列文件中的内容通过文中的规范性引用而构成本文件必不可少的条款。其中，注日期的引用文件，仅该日期对应的版本适用于本文件；不注日期的引用文件，其最新版本（包括所有的修改单）适用于本文件。

GB/T 16263.4-2015 信息技术 ASN.1编码规则 第4部分：XML编码规则

3 术语和定义

下列术语和定义适用于本文件。

3.1

绿色智能船舶协同研发平台模型单元 green& intelligent ship collaborative r&d platform model unit (GISSMU)

用于船舶系统模型的，由多个存储具有预定义结构的文件组成的 ZIP 文件。文件内容详见附录 A.1，GISSMU 文件生成流程详见附录 A.2 及附录 B 示例。

3.2

协同仿真 co-simulation

一种将不同工具开发的模型进行联合仿真的方法。

3.3

功能标志 capability flag

用于指示导入工具GISSMU支持的可选功能。

3.4

导入工具 importer

导入或加载一个或多个GISSMU的软件。也称为仿真环境、环境、调用环境、（协同）仿真算法、目标平台、目标环境、集成器。

3.5

模型描述文件 model description file

一个名为modelDescription.xml的XML文件，提供模型的所有属性的描述（例如输入/输出变量）。

3.6

仿真程序 simulation program

用于开发和/或求解仿真模型的软件。该软件包含一个求解器，可包含一个用户界面以及后处理方法。

3.7

通信点 communication points

在协同仿真环境中，用于导入工具和GISSMU之间交换数据的时间网格。

3.8

通信步长 communication step size

两个相继通信点之间的时间间隔。

3.9

用户界面 user interface

仿真程序的一部分，提供用户对仿真的控制并允许观看结果。

3.10

接口函数 interface functions

FMI标准中定义的标准化API，用于实现GISSMU与仿真环境间的交互，涵盖模型初始化、执行、状态管理及数据读写等功能。

3.11

接口属性 interface properties

FMI标准中定义GISSMU接口行为、元数据及交互规则的核心配置参数，用于确保模型在跨平台协同仿真中的兼容性与功能性。

4 缩略语

下列缩略语适用于本文件。

API：应用程序编程接口（Application Programming Interface）

DLL：动态链接库（Dynamic Link Library）

FMI：功能模型接口（Functional Mock-up Interface）

GISSMU：绿色智能船舶协同研发平台模型单元（Green Intelligent ship simulation model unit）

MPI：消息传递接口（Message Passing Interface）

SO：共享对象（Shared Object）

TCP/IP：传输控制协议/网际协议（Transmission Control Protocol/Internet Protocol）

XML：可扩展标记语言（extensible markup language）

5 GISSMU 生成的通用要求

5.1 核心元数据

5.1.1 模型描述文件

5.1.1.1 模型定义

GISSMU应包含一个独立定义的模型描述文件，该文件采用FMI标准中规定的XML格式，用于准确描述模型本体及其相关元数据。模型描述文件应作为模型的一部分独立存在，与模型代码分离存储，应完整列出模型全部变量、输入输出接口、参数信息及其他元数据，便于平台统一解析与管理。

a) 在 FMI 框架中，所有与 GISSMU 核心功能相关的静态信息均应存储于该 XML 文件。文件应遵循 FMI-3 标准定义的 XML 模式文件`fmi3ModelDescription.xsd`，其主要组成内容包括：

- 模型属性（Attributes）；
- 默认实验设置（DefaultExperiment）；
- 模型变量（ModelVariables）；
- 单位定义（UnitDefinitions）；
- 类型定义（TypeDefinitions）；
- 日志类别（LogCategories）；
- 模型结构（ModelStructure）。

其中，模型属性（Attributes）、模型变量（ModelVariables）及模型结构（ModelStructure）为必选项，其他字段为可选项，可根据建模需求灵活配置。

b) XML 文件中的模型属性应至少包括以下信息，以实现模型的唯一识别和独立定义：

- FMI 版本（fmiVersion）；
- 模型名称（modelName）；
- 模型标识符（modelIdentifier）；
- 全局唯一标识符（guid）；
- 连续状态数量（numberOfContinuousStates）；

- 事件指示器数量 (numberOfEventIndicators)；
- 版本号 (version)；
- 生成工具 (generationTool)；
- 生成时间 (generationDateAndTime)；
- 变量命名约定 (variableNamingConvention)；
- 模型描述 (description)。

注：`version`、`generationTool`、`generationDateAndTime`、`variableNamingConvention`、`description` 在技术实现上可作为可选项，如存在应如实填写。

模型描述文件生成后应保持只读，不允许在 GISSMU 归档包内部对该文件进行修改。模型变量初始值发生改动将直接影响仿真计算准确性，应保证 XML 文件内容与 GISSMU 的二进制代码或源代码保持一致，保障模型行为的稳定性和一致性。

5.1.2 XML 数据类型与映射

FMI 使用的数据类型应与 C 语言、FMI 标准对应类型匹配，映射关系见表 1。

表1 XML 数据类型与映射表

XML	映射到C	映射到FMI
double	double	fmi3Float64
single	float	fmi3Float32
byte	int8_t	fmi3Int8
unsignedByte	uint8_t	fmi3UInt8
short	int16_t	fmi3Int16
unsignedShort	uint16_t	fmi3UInt16
int	int32_t	fmi3Int32
unsignedInt	uint32_t	fmi3UInt32
long	int64_t	fmi3Int64
unsignedLong	uint64_t	fmi3UInt64
boolean	bool	fmi3Boolean
string	char*	fmi3String
normalizedString	char*	fmi3String
hexBinary	uint_8*	fmi3Binary
dateTime	tool specific	not defined

5.1.3 name 属性的唯一性

在`modelDescription.xml`文件中，所有`name`属性（例如变量的名称）应是唯一的，且不能为空字符串。

5.1.4 有序列表与属性集合

FMI的架构基于有序列表，在模型的连续状态求解过程中，元素的顺序应保留。这意味着解析器在处理`modelDescription.xml`文件时，应确保列表项的顺序一致，以保证仿真过程的正确性。

5.2 文件结构与编码

5.2.1 文件结构

5.2.1.1 必需文件与目录

GISSMU 分发包应配套下列文件与目录，用于定义模型信息、存放二进制实现及说明外部依赖：

- a) 应包含 modelDescription.xml，描述 GISSMU 的模型结构、变量、接口等。
- b) binaries/目录（二进制 GISSMU 必需）：
 - 1) 平台子目录：按<arch>-<sys>格式命名（如 x86_64-windows），包含对应平台的共享库（.dll、.so、.dylib）；
 - 2) 外部依赖记录：若依赖外部库，应在 documentation/externalDependencies.{txt|html} 中记录。

- c) GISSMU 依赖外部资源（共享库、服务器）时，应提供 documentation/externalDependencies. {txt|html} 文件。

5.2.1.2 可选目录与规则

GISSMU 分发包可增设下列可选目录，各目录配套文件与约束规则如下：

- a) documentation
 - 可选文件：index.html（文档入口）、diagram.png/svg（模型图表）及其他文档；
 - 条件规则：若提供 diagram.svg，应同时提供 diagram.png。
- b) licenses
 - 入口文件：存在该目录时，应包含 license.spdx、license.txt 或 license.html；
 - 建议：应包含全部开源组件许可证信息。
- c) sources（源代码 GISSMU）
 - 构建文件：应包含 buildDescription.xml，定义平台构建配置；
 - 头文件：不包含标准头文件（如 fmi3Functions.h），该文件由导入工具提供。
- d) terminalsAndIcons
 - 可选内容：图标文件（icon.png/svg）、终端描述文件（TerminalsAndIcons.xml）；
 - 规则：若提供 icon.svg，应同时提供 icon.png。
- e) resources
 - 用途：存放运行时资源（数据表、配置文件），通过 resourcePath 访问；
 - 限制：GISSMU 实例运行过程中不可修改、删除资源文件。
- f) extra
 - 扩展数据：子目录采取反向域名格式命名，例如 extra/com.example/；
 - 允许修改：其他工具可修改此目录内容，修改不影响 GISSMU 本身有效性。

5.2.2 XML 文件的编码

所有 XML 文件（如 modelDescription.xml）第一行应声明编码。

示例：<?xml version="1.0" encoding="UTF-8"?>

依据 GB/T 16263.4-2015，XML 文件应采用 UTF-8 编码。XML 应具备完整结构，包含 XML 声明、根元素及子元素；可添加注释，不宜过度注释；采用缩进格式化，推荐两个空格缩进。

5.2.3 辅助架构文件

GISSMU 可配套下述辅助架构文件，分别用于定义图形终端信息与源代码构建配置：

- a) 图形表示与终端分组：GISSMU 可包含可选的 terminalsAndIcons/terminalsAndIcons.xml 文件，用于定义模型变量图形表示和终端分组，文件应遵循 fmi3TerminalsAndIcons.xsd 模式文件。
- b) GISSMU 源代码构建信息：GISSMU 附源代码时，应提供 buildDescription.xml，遵循 fmi3BuildDescription.xsd 模式文件。

5.3 接口与线程安全

5.3.1 C-API 实现的要求

GISSMU 的 FMI C-API 实现应遵守下列约束：

- a) FMI 实例函数不强制线程安全。多线程环境调用同一 GISSMU 实例函数时，由仿真环境保证无竞态条件，GISSMU 内部不提供线程同步机制；
- b) fmi3LogMessageCallback、fmi3LockPreemptionCallback、fmi3UnlockPreemptionCallback 回调函数不得调用 GISSMU 内部接口，否则产生未定义行为；
- c) fmi3IntermediateUpdateCallback 和 fmi3ClockUpdateCallback 可在确定状态下调用 FMI 函数，例如时间步推进过程更新时钟或中间状态；

- d) FMI 函数不得修改跨进程 / 跨线程全局配置（CPU 浮点控制寄存器等）；如修改线程局部浮点设置，函数返回前必须恢复原始状态；
- e) FMI 函数入参禁止传入 NULL，标准明确允许除外（例如`resourcePath`参数可为 NULL 代表无资源路径）；实现应做参数防御性检查，避免空指针崩溃；
- f) FMI 不提供运行时平台抽象层，GISSMU 访问操作系统资源（内存、文件、网络）应做容错；目标平台不支持对应资源时，记录错误并返回`fmi3Error`状态码。

5.3.1 状态机和语义

FMI 标准通过严格状态机保障 GISSMU 仿真时序正确性，状态及转换规则如下：

- a) 超级状态（可设置 GISSMU 状态）仅允许在实例化和初始化模式下调用状态修改接口。从实例化状态进入初始化模式前，需完成所有必要参数的设置。
- b) 实例化状态：调用 `fmi3Instantiate` 创建 GISSMU 实例，分配内存、加载模型描述文件。仅允许基础查询接口、实例化相关函数；禁止操作模型变量、触发计算。XML 解析等资源加载失败应返回 `fmi3Error` 并释放实例。
- c) 初始化模式：实例化完成后调用 `fmi3EnterInitializationMode` 进入；任务包括设置变量初始值、校验模型一致性；所有输入变量应在初始化完成前赋值，否则返回错误；调用 `fmi3ExitInitializationMode` 退出初始化，进入已初始化超级状态。
- d) 已初始化超级状态：包含事件模式、配置模式等子状态；支持时间推进、事件处理；禁止修改静态元参数，可调整动态可配置参数。 e)
- e) 事件模式：由离散事件（时钟、阈值越界）或外部调用 `fmi3UpdateDiscreteStates` 触发；检测事件指示器、处理状态跳变，返回事件处理结果；事件处理应在同一个仿真步内完成，禁止跨步异步执行。
- f) 配置模式与重新配置模式：
 - 1) 配置模式：通过 `fmi3EnterConfigurationMode` 显式进入，允许动态调整模型结构、变量连接；
 - 2) 重新配置模式：仿真运行中修改参数 / 结构后，调用 `fmi3Reset` 重新初始化，保证状态一致性；
 - 3) 配置操作应原子执行，避免部分更新造成状态冲突。
- g) 终止状态：触发条件为调用 `fmi3Terminate` 或发生不可恢复错误；释放全部动态资源，记录终止原因；终止后实例不可复用，需要重新实例化。
- h) 状态转换与接口时序：
 - 1) 严格顺序转换，禁止跨状态跳转；
 - 2) 事件模式禁止调用初始化接口；终止状态除 `fmi3FreeInstance` 外其余接口返回 `fmi3Fatal`；
 - 3) 状态操作失败时，GISSMU 应回滚至上一稳定状态并输出日志

5.4 二进制与兼容性

5.4.1 功能标志支持

GISSMU 生成时应包含 FMI 标准规定的功能标志，指示所支持可选功能（事件处理、动态状态选择等）；导入工具读取标志调整仿真策略。

5.4.2 版本控制与兼容性

GISSMU 应明确标注 FMI 版本号（FMI3.0）；生成方应评估模型跨版本 FMI 的兼容能力。

5.5 扩展与调试

5.5.1 注解机制

允许在 XML 元素中增加`Annotation`扩展元素；每个`Annotation`必须设置`type`命名空间属性；基础 FMI 实现可安全忽略注解，不影响仿真运行。

二进制库应与目标操作系统平台兼容；GISSMU 针对平台编译，应可在目标平台正常调用接口运行。

5.5.2 错误处理与调试信息

GISSMU 应具备错误处理机制，故障场景输出调试信息，便于定位仿真问题。

5.5.3 多语言支持

GISSMU 可提供 C、C++、Python 等语言接口层，适配不同仿真环境。

6 GISSMU 生成要求

6.1 GISSMU 代码

6.1.1 代码生成要求

生成的GISSMU ZIP存档应遵循ZIP规范，使用正斜杠作为路径分隔符。ZIP存档应不包含前导前缀的条目。

示例：modelDescription.xml 和 binaries/x86_64-windows/myGISSMU.dll。

文档和资源代码生成应满足以下要求：

- 应包含图标信息，这些信息应包含在 GISSMU 文件中；
- 应包含标准的元数据属性，如描述、作者、版本、版权、许可证、生成工具和生成日期等；
- 应包含所有必要的文档，以使用户能够理解和运行 GISSMU。

GISSMU 可同时提供 32 位、64 位二进制，支持多平台二进制共存。应记录编译器、版本、编译标志。

FMI3.0 环境下，源代码使用`fmi3Functions.h`宏控制函数前缀；导出接口仅使用头文件宏生成的 API 入口名称。

尽量减少额外动态库依赖；存在外部依赖时应记录。GISSMU 函数不应修改进程全局环境设置。

6.1.2 打包和代码生成加载

GISSMU 应按照 FMI 标准完成代码生成、编译、打包与加载，打包交付物、命名规则、编译配置、校验要求应符合下列规定：

- 采用符合 FMI 标准的建模工具生成 GISSMU 源代码或二进制实现。
- GISSMU 归档包包含：XML 模型描述文件、编译后二进制、可选源代码、运行必需资源。
- 模型标识符用于构造 C 函数名、库文件名，必须符合 C 标识符命名限制；静态库文件名应与`modelIdentifier`一致；API 函数名采用模型标识符 + 函数名，下划线分隔。
- 编译前配置工程环境（包含目录、链接库、平台工具集）；实现 FMI3 全部必要回调函数，构建模型内部数据结构。
- GISSMU 包含 C 源代码或静态库时，编译前定义宏`FMI3\_FUNCTION\_PREFIX`等于`modelIdentifier`取值。
- 代码应明确定义参数、变量、初始值、边界条件，保障数值计算精度与效率；编译使用记录的编译标志。
- 完成编译后生成`modelDescription.xml`元数据文件。生成后应在不同平台开展测试，设置初始参数运行仿真，校验结果。打包完成后交付 GISSMU，配套文档说明。

6.1.3 参数和变量处理

GISSMU 应对模型参数与变量进行规范化定义、访问控制与生命周期管理，应符合下列规定：

- 区分输入参数（初始、边界条件）、输出结果变量；明确变量数据类型、值域、物理单位。
- 明确变量作用域（局部 / 全局）；多线程 / 并行计算场景重点管控全局变量访问。
- 变量使用前完成初始化，避免未定义行为；支持用户不修改代码即可配置参数；每个参数提供文档说明物理含义、取值范围。
- 增加参数合法性校验逻辑；非法参数输出明确错误信息，可提供合理回退策略。
- 变量命名避免混淆缩写；代码增加注释说明；文档完整描述全部参数变量。
- 动态分配内存要管理生命周期，防止内存泄漏；并行环境下共享变量访问应做线程安全处理。

- g) 模型状态可序列化 / 反序列化，支持状态保存恢复。优先使用代码生成工具完成类型校验、内存管理。

6.2 变量

6.2.1 GISSMU 所需变量要求

GISSMU 变量按因果性分为参数、输入、输出、局部变量四类，定义如下：

- 参数：仿真开始前设置，仿真过程不变；`modelDescription.xml` 中 `causality="parameter"`；
示例：`<ScalarVariable name="parameter1" valueReference="1" causality="parameter"></ScalarVariable>`
- 输入：外部驱动信号，仿真过程可动态改变；`causality="input"`；
示例：`<ScalarVariable name="input1" valueReference="2" causality="input"></ScalarVariable>`
- 输出：模型计算输出结果；`causality="output"`；
示例：`<ScalarVariable name="output1" valueReference="3" causality="output"></ScalarVariable>`
- 局部变量：模型内部中间状态，外部不可访问；`causality="local"`。
示例：`<ScalarVariable name="localVariable1" valueReference="4" causality="local"></ScalarVariable>`

6.2.2 变量属性要求

GISSMU 变量应满足下列属性约束：

- 名称：不含空格与特殊字符，在 `modelDescription.xml` 中唯一；
- 值引用：唯一 32 位无符号整数；
- 描述：说明物理意义、单位、取值范围；
- 因果性：明确 parameter / input / output / local；
- 可变性：标识连续 / 离散变化；
- 初始值：状态类变量应提供 start 初始值。

6.2.3 变量类型要求

GISSMU 支持的变量类型及标签约定如下：

- 实数：`<Real>` 标签，双精度浮点数；
- 整数：`<Integer>` 标签，支持 32/64 位有符号整数；
- 布尔：`<Boolean>` 标签，true/false；
- 字符串：`<String>` 标签，UTF-8 编码文本。

6.3 GISSMU 结构和文件说明

6.3.1 模型描述文件要求

模型描述文件应包含：

- 模型属性信息（FMI 版本号、模型名称、GUID 等）；
- 模型变量信息（变量名称、类型、初始值等）；
- 模型结构定义（输出、状态导数、初始化未知量等）。

6.3.2 模型属性信息要求

模型属性信息应满足以下要求：

- 明确指定 FMI 版本号；
示例：`<fmiVersion>3.0</fmiVersion>`
- 提供模型的名称，且名称应唯一；
示例：`<modelName>MyModel</modelName>`
- 提供一个全局唯一的标识符，用于区分不同的 GISSMU；
示例：`<guid>{12345678-1234-1234-1234-123456789012}</guid>`
- 指定变量的命名约定，如 structured 或 flat；
示例：`<variableNamingConvention>structured</variableNamingConvention>`
- 指定模型中事件指示器的数量。
示例：`<numberOfEventIndicators>2</numberOfEventIndicators>`

6.3.3 共享库文件

共享库（Windows dll / Linux so）为模型二进制实现。生成规范：

- a) 明确共享库功能，设计 API 接口；
- b) 使用 C/C++ 编写源代码，做好注释文档；
- c) 使用 GCC、Clang、MSVC 等编译器编译生成共享库；
- d) 嵌入版本信息，采用语义化版本；
- e) 单元测试、集成测试，验证跨平台兼容性；
- f) 配套 API 文档、调用示例。

6.4 接口要求

6.4.1 公共接口

必须实现以下接口：

- a) fmi2Instantiate：实例化 GISSMU 模型；
- b) Fmi3SetupExperiment：设置仿真参数（开始时间、结束时间、容差等）；
- c) Fmi3EnterInitializationMode/fmi3ExitInitializationMode：进入/退出初始化模式；
- d) Fmi3Terminate/fmi2FreeInstance：终止仿真并释放资源；
- e) Fmi3SetXXX/fmi3GetXXX（XXX 为 Real/Integer/Boolean/String）：设置/获取模型变量值；

6.4.2 Co-Simulation 类型专用接口

应实现以下接口：

- a) Fmi3DoStep：执行单步仿真；
- b) Fmi3GetStatus：获取仿真状态。

6.4.3 求解器接口要求

应调用以下接口：

- a) Fmi3SetTime：设置仿真时间；
- b) Fmi3GetDerivatives：获取导数；
- c) Fmi3SetContinuousStates：设置连续状态。

6.4.4 联合仿真平台功能接口要求

6.4.4.1 模型管理应支持以下功能：

- a) 导入/导出 GISSMU 模型；
- b) 可视化展示模型变量及连接拓扑；
- c) 提供模型重置（fmi3Reset）和日志记录功能。

6.4.4.2 仿真流程控制应支持以下功能：

- a) 启动、暂停、步进、停止仿真；
- b) 多线程调度，确保实时性与数据同步。

6.4.4.3 数据可视化应支持实时绘制 GISSMU 输出变量曲线；支持多窗口对比分析。

6.4.5 接口调用时序要求

接口调用应遵循FMI标准中规定的时序流程，确保模型运行的正确性。典型的调用顺序如下：

6.4.5.1 模型实例化阶段，本阶段完成模型实例创建与仿真实验前置配置。

- a) 调用 fmi2Instantiate 创建模型实例。
- b) 可选：调用 fmi3SetupExperiment 配置仿真时间与容差。

6.4.5.2 初始化阶段，本阶段完成模型初始化模式切换与初始变量赋值。

- a) 调用 fmi3EnterInitializationMode 进入初始化模式；
- b) 设置初始变量（通过 fmi3SetXXX）；
- c) 调用 fmi3ExitInitializationMode 完成初始化。

6.4.5.3 仿真执行阶段 (Co-Simulation 类型), 本阶段开展协同仿真迭代计算, 支持变量动态读写与状态查询。

- a) 多次调用 fmi3DoStep 执行单步仿真;
- b) 可调用 fmi3GetStatus 获取当前仿真状态;
- c) 仿真过程中可动态调用 fmi3SetXXX 和 fmi3GetXXX 设置/获取变量值。

6.4.5.4 仿真结束阶段, 本阶段执行仿真终止与模型资源释放。

- a) 调用 fmi3Terminate 终止仿真;
- b) 调用 fmi2FreeInstance 释放模型资源。

6.5 GISSMU 分布式部署

6.5.1 GISSMU 分布式部署规范要求

分布式部署可选用 TCP/IP 或 MPI 通信协议, 应优化网络降低通信延迟 (面向实时仿真)。

- a) 各部署节点操作系统、仿真工具链版本尽量保持一致;
- b) 使用兼容版本 GISSMU, 接口定义保持一致;
- c) 可使用 Docker、Kubernetes、Ansible 等工具完成部署配置管理。

6.5.2 GISSMU 部署要求

GISSMU 部署要求如下:

- a) GISSMU 归档包 binaries 目录下按<arch>-<sys>存放对应平台二进制库;
- b) 查阅 documentation/externalDependencies.{txt|html}, 补齐全部外部依赖;
- c) 保证 resources 目录资源文件运行时可访问;
- d) 将 GISSMU 压缩包解压至目标主机, 选择匹配平台二进制;
- e) 配置操作系统环境变量 (Windows PATH、Linux LD_LIBRARY_PATH);
- f) 使用系统动态库加载 API 加载二进制库;
- g) 使用 FMI3-API 实例化 GISSMU;
- h) 通过 FMI 接口执行仿真、读写变量。

7 验证和测试要求

7.1 GISSMU 验证规则

7.1.1 变量组验证规则

变量组验证规则如下:

- a) 变量组名称唯一, 每个<VariableGroup>元素应拥有唯一的名称, 在给定的 GISSMUModelDescription.xml 中没有重复的变量组名称. 变量名称唯一, 所有<Variable>元素也应有唯一的名称。
- b) 变量类型一致, 对于定义为特定类型的变量组 (AngularAccelerationType、AngularDisplacementType 等), 其中的所有变量应具有相同的因果关系、数据类型和基本单位。变量数量限制, 某些变量组 (Force、LinearVelocity) 应包含 1 到 3 个变量, 且所有变量应具有相同的因果关系。
- c) 变量组连接只能连接两个同类型但具有相反因果性的变量组。

示例: 一个<LinearMechanicalPort>只能与另一个<LinearMechanicalPort>相连, 并且它们的内部变量应该具有相反的因果关系。

- d) 变量组内部结构, 对于复合变量组 (如 LinearMechanicalPort), 其内部应包含一个 Force 变量组和一个 LinearVelocity 变量组, 且这两个变量组应包含相同数量的变量, 且变量的因果关系相反。

7.1.2 变量连接验证规则

变量连接验证规则如下:

- a) 唯一性，每个变量连接应具有唯一的 IRI（国际资源标识符），并且不能与其他个体相同。以及类型匹配，变量连接只能连接两个具有相同数据类型、相反因果关系和相同基本单位的变量。
- b) 单位兼容性，变量连接中的两个变量应具有相容的单位，
- c) 力不能直接连接到电压上。
- d) 因果关系，连接的两个变量之间应具有相反的因果关系，以确保信号可以正确地从一个方向传递到另一个方向。变量顺序，对于有序变量组（LinearVelocity），连接时应保持变量的顺序一致。

7.1.3 配置验证规则

7.1.3.1 模型器配置验证

模型器配置验证应满足一下要求：

- a) 模拟器名称唯一性，确保所有模拟器都有一个独一无二的名字，以避免混淆。
- b) 模拟器接口一致性，验证模拟器的输入和输出接口与 ModelDescription.xml 中定义的一致。
- c) 模拟器源文件路径，确保每个模拟器的源文件路径是正确的，并且文件存在且可访问。
- d) 模拟器版本兼容性，确保所有模拟器的版本与 OspModelDescription.xml 中定义的版本一致。

7.1.3.2 连接配置验证

连接配置验证应满足一下要求：

- a) 连接正确性，使用 VariableGroupConnection 设置的连接的变量组是同一类型，且内部变量具有相反的因果性。
- b) 连接的唯一性，每个 VariableGroupConnection 应具有唯一的名称，在 SystemStructure.xml 中没有重复的连接名称。
- c) 正确的连接顺序，确保连接的变量组在连接中按正确的顺序排列。
- d) 确保连接完整性，每个连接应包含所有必要的信息。

7.1.3.3 仿真参数配置验证

仿真参数配置验证应满足一下要求：

- a) StartTime 和 BaseStepSize 等仿真参数在 OspSystemStructure.xml 中应正确设置。
- b) 验证仿真算法（fixedStep）在 OspSystemStructure.xml 中正确设置，并且与所有模拟器的算法兼容。
- c) 仿真结束时间在 OspSystemStructure.xml 中正确设置，且大于开始时间。

7.1.3.4 配置文件完整性

配置文件完整性应满足一下要求：

- a) ModelDescription.xml 和 SystemStructure.xml 文件的结构应符合 XML Schema 定义，使用 ModelDescription-1.0.0.xsd 和 SystemStructure-1.0.0.xsd 进行验证。
- b) 所有必要的信息（变量组、变量、单位、连接等）在配置文件中应完整且正确。

7.1.4 实施验证

7.1.4.1 填充本体

填充本体应满足如下要求：

- a) 使用 XML 解析器读取 ModelDescription.xml 和 SystemStructure.xml 文件。解析文件内容，提取模型描述、变量组、变量、连接关系等信息。
- b) 根据提取的信息，在 OWL 2 本体中创建相应的个体。例如，为每个模拟器创建一个 Simulator 个体，为每个变量组创建一个 VariableGroup 个体，为每个变量创建一个 Variable 个体。
- c) 为每个个体设置名称、数据类型、单位、因果关系等必要属性。
- d) 根据 SystemStructure.xml 中的连接信息，创建 VariableConnection 和 VariableGroupConnection 个体。设置连接关系的属性。

- e) 如连接的模拟器名称、变量组名称、变量名称等，确保连接关系的因果关系、数据类型和单位一致。
- f) 强调在整个验证过程中保持详细的日志记录的重要性，以便于问题追踪和后续审查。
- g) 建立从验证结果到开发者之间的快速反馈通道，确保及时修正发现的问题。

7.1.4.2 分类验证

分类验证应满足如下要求：

- a) 使用 OWL API 加载填充好的本体文件。
- b) 使用 OWL Reasoner (HermiT 或 Pellet) 对本体进行分类，分类过程会检查模型配置和连接配置是否违反了任何规则。
- c) 在分类完成后，从本体中检索出被分类到错误类的个体，详见附录 C。
- d) 检索出被分类到 InvalidVariableConnection 或 InvalidVariableGroupConnection 类的个体。
- e) 根据检索到的错误类，生成详细的错误消息，错误消息应包含具体的错误类型、涉及的个体名称和相关属性。

7.1.5 测试要求

7.1.5.1 初始条件验证

初始条件验证应满足如下要求：

- a) 模型全部初始变量、初始时间设置正确；
- b) 初始条件逻辑自洽，无冲突。

7.1.5.2 仿真执行过程验证

仿真执行过程验证应满足如下要求：

- a) 多子模型协同仿真时，通信步长配置合理；
- b) 时间同步机制有效，每个通信点完成全部子模型状态更新；
- c) 变量运行值处于合理值域，无 NaN、无穷等异常值；
- d) 变量变化率合理，无不合理突变。

7.1.5.3 仿真结果验证

仿真结果验证应满足如下要求：

- a) 仿真结果与预期结果比对，误差在可接受范围；
- b) 多子模型联合仿真时，物理守恒特性（能量、功率传输）应符合物理规律。

7.1.5.4 结果复核

结构复核应满足如下要求：

- a) 可开展第三方独立复核；
- b) 关键指标提供结果解释。

7.1.5.5 文档和版本管理

文档和版本管理应满足如下要求：

- a) 保存测试文档：模型配置、仿真设置、结果分析报告；
- b) GISSMU 模型、配套文档执行版本控制，支持回溯。

附录 A (资料性) 术语和定义补充内容

A.1 GISSMU 文件内容

A.1.1 模型描述文件

XML 文件描述了 GISSMU 的接口信息、变量、输入输出、参数以及其他元数据。它是 GISSMU 的核心，通过该文件定义了 GISSMU 与不同系统之间的交互。

A.1.2 二进制库文件

GISSMU 的二进制实现通常以共享库（如 Windows 上的 DLL，Linux 上的 .so 文件）形式提供，这些文件包含了具体的模型代码，二进制库可包含求解器。

A.1.3 资源文件

这些文件包括 GISSMU 运行时所需的其他辅助文件，例如数据文件、配置文件或脚本。资源文件通常存放在 GISSMU 的 resources 目录下。

A.1.4 可选的源代码文件

某些 GISSMU 可能附带源代码，允许用户在不同平台上重新编译 GISSMU。这些文件通常会包含在 sources 目录下。

A.2 GISSMU 文件生成流程

A.2.1 模型开发

A.2.1.1 在支持 FMI 标准的建模工具中开发模型（如 Modelica、Simulink、Matlab 等）。

A.2.1.2 模型可以是基于物理的数学模型，通常由微分方程（ODEs 或 DAEs）描述，同时可能包含离散事件逻辑。

A.2.2 配置 GISSMU 导出参数

A.2.2.1 设置模型参数：配置模型的关键参数，包括变量范围、初始条件、事件处理方法等。

A.2.2.2 定义接口变量：定义输入、输出以及中间变量。这些变量在 GISSMU 的 XML 文件中描述，并用于模型交互。

A.2.3 生成 FMI 描述文件（.XML 文件）

A.2.3.1 建模工具会根据模型的配置生成符合 FMI 标准的 XML 文件，包含以下内容：

- 模型元信息：模型名称、作者、生成日期、版本等。
- 变量定义：变量名、数据类型（Real、Integer、String、Boolean）、初始值等。
- 输入输出描述：指定哪些变量是输入变量、输出变量或状态变量。
- 时间事件和状态事件的信息（如果存在）。
- 其他元数据信息（如数值求解器信息等）。

A.2.3.2 生成动态链接库（DLL/SO 文件）

A.2.3.2.1 将模型转换为 C 代码：建模工具会将模型的数学描述（ODE、DDE 或离散逻辑）自动生成相应的 C 代码。

A.2.3.2.2 编译 C 代码为共享库（DLL 或 SO 文件）：编译后的动态链接库实现了 FMI 标准定义的 C 接口函数，如：fmiInstantiateModel、fmiInitialize、fmiSetTime、fmiGetDerivatives、fmiTerminate 等。

A.2.3.3 打包为 GISSMU 文件

- 以下内容打包为一个 .GISSMU 压缩文件（ZIP 格式，后缀为 .GISSMU）
- 共享库文件（DLL/SO）：实现 GISSMU 的功能逻辑。
 - XML 文件：描述模型的变量、事件、初始条件等元信息。
 - 可选资源文件：包括文档（如 README）、表格数据、位图文件等，帮助用户理解模型或提供附加数据支持。

A.2.3.4 测试生成的 GISSMU

- A.2.3.4.1 使用支持 FMI 标准的仿真环境（如 JModelica.org、OpenModelica、PyFMI 等）测试 GISSMU。
- A.2.3.4.2 验证 GISSMU 的功能是否正确，包括接口的输入输出、事件处理、仿真结果的准确性等。

A.3 协同仿真计算流程

A.3.1 初始化阶段

- A.3.1.1 所有仿真工具为开始协同仿真做好准备。
- A.3.1.2 主控模块（master）获取从属模块（slaves）的属性以及连接图。
- A.3.1.3 初始化从属模块和模型，并设置相关参数。
- A.3.1.4 建立主控模块与从属模块之间的通信链接。
- A.3.1.5 主控模块根据从属模块的功能标志、连接图以及用户输入选择合适的算法。

A.3.2 仿真阶段

- A.3.2.1 主控模块通过逐步求解主控子区间（称为通信步）来迫使从属模块完成从起始时间到终止时间的仿真。通信步的边界称为通信点。
- A.3.2.2 在事件迭代的情况下，通信步长可以为零。
- A.3.2.3 仿真对各子系统独立执行，子系统之间的数据交换仅限于通信点。
- A.3.2.4 在模拟一个子区间之前，从属模块从主控模块接收输入值及其相对于时间的导数（如适用），以及通信步长。
- A.3.2.5 完成通信步后，主控模块接收从属模块的输出值及其可能的导数，同时还需要传递从属模块的状态信息。如果从属模块仿真失败，则需要进一步通信。

A.3.3 关闭阶段

主控模块停止整个仿真过程，释放内存，并终止、重置或关闭从属模块。

A.4 变量类型

表A.1 变量类型

变量类型	描述
Real（实数）	用于表示连续的数值，如温度、速度、压力等。Real类型可以进一步细分为不同的宽度，如 fmiFloat32和fmiFloat64，分别对应32位和64位浮点数。
Integer（整数）	用于表示整数值，如计数、索引或布尔逻辑状态。Integer类型也可以有不同的宽度，如fmiInt8、fmiUInt8、fmiInt16、fmiUInt16、fmiInt32和fmiUInt32。
Boolean（布尔）	表示逻辑状态，通常只有两个可能的值：true或false。在FMI中，布尔类型通常用fmiBoolean表示。
String（字符串）	用于表示文本数据，如标签、名称或描述。字符串类型用fmiString表示。
Enumeration（枚举）	用于表示一组预定义的值。枚举类型允许变量取自一个有限的值集合，这些值通常在模型描述中列出。
Clock（时钟）	特殊类型的变量，用于表示仿真中的事件或时间点。时钟变量通常是标量，并且它们的值通常由仿真环境控制。
Binary（二进制）	用于表示二进制数据流，如文件或内存缓冲区的内容。二进制数据类型用fmiBinary表示。
Complex（复数）	虽然在FMI标准中不是核心类型，但一些扩展可能支持复数类型，用于表示具有实部和虚部的值。

A.5 变量属性

表A.2 变量属性

属性	描述
Value Reference	每个变量都有一个唯一的值引用，它是一个整数，用于在FMI函数调用中标识变量。
Name	变量的名称是一个字符串，用于在模型描述文件中标识变量。
Variable Type	变量可以是多种类型之一，包括但不限于Real、Integer、Boolean、String等。
Causality	定义了变量的因果关系，指出变量是作为输入(input)、输出(output)还是参数(parameter)。
Variability	描述变量值在仿真过程中的变化情况，如constant (常数)、tunable (可调)、discrete (离散) 等。
Start	为变量指定一个初始值，这个值在仿真开始时被使用。
Min/Max	定义了变量的取值范围，用于指导仿真环境或进行验证。
Unit	提供变量的物理单位，例如seconds、meters、Newtons等。
DisplayName	提供变量的友好名称，通常用于图形用户界面显示。
Description	变量的详细描述，可能包括其在模型中的用途、约束或其他相关信息。

A.6 模型描述文件元素

表A.3 模型描述文件元素

元素	描述
<ModelExchange>	GISSMU 基于 FMI 进行模型交换
<CoSimulation>	GISSMU 基于 FMI 进行协同仿真
<ScheduledExecution>	GISSMU 基于计划执行的 FMI
<UnitDefinitions>	单位和显示单位定义的列表，这些定义在 XML 元素<ModelVariables> 中使用。
<TypeDefinitions>	在<ModelVariables> 中使用的类型定义的列表。
<LogCategories>	日志类别列表，可以设置这些类别来定义 GISSMU 支持的日志信息。
<DefaultExperiment>	导入器算法的默认设置，例如停止时间和相对容差。
<ModelVariables>	可通过 GISSMU 函数访问的 GISSMU 的所有变量的列表。
<ModelStructure>	定义模型的结构。特别是，输出、连续时间状态、时钟状态、初始未知数（和事件指示符的有序列表在此处定义。此外，可以选择为输出、连续时间状态和初始未知数定义未知数与已知数的依赖关系。
<Annotations>	顶级元素的可选注释。

A.7 模型变量属性

表A.4 模型变量属性

属性或元素	描述
name	使用属性 declaredType 引用的变量类型的唯一名称。此属性必须与变量的所有属性不同（如果使用相同的名称，那么在使用 Modelica 语言的环境中导入 GISSMU 时，这几乎总是会出现问题，其中类型名称不能用作实例名称）。
description	描述变量类型含义的可选描述字符串。
quantity	变量的物理量。
unit	引用具有相同 的变量的 Unit，该变量必须存在。没有 unit 的类型定义不能具有属性。
displayUnit	引用 a 的变量的默认显示单位，该变量必须存在于 of 下。到 的转换由元素 ，如果已定义，则还必须定义 a。
mimeType	指示作为二进制文件传递的数据类型。默认为，这是不特定的。实现可以使用此信息来提供有关有效/有用连接的指导。
maxSize	指示作为二进制文件传递的数据的最大大小。
relativeQuantity	如果此为属性，则和必须忽略。
min	变量的最小值（variable value）。如果未定义，则最小值是计算机上可以表示的最大负数。定义 GISSMU 设计运行的区域。
max	变量的最大值（variable value）。如果未定义，则 maximum 是计算机上可以表示的最大正数。定义 GISSMU 设计运行的区域。
nominal	变量的标称值。如果未定义且没有有关标称值的其他信息可用，则假定为。对于数组变量，此标称值适用于数组的所有元素。
unbounded	表示在时间积分期间，变量获得的值远大于其标称值。

表A. 4 模型变量属性（续）

Item	必须具有非空元素序列。
------	-------------

A. 8 接口函数

表A. 5 接口函数

函数名	主要功能
fmi3Instantiate ()	实例化 GISSMU 模型，后续对实例的操作便是对GISSMU模型的操作
fmi3SetupExperiment ()	设置 GISSMU 模型仿真参数，包括当前模型的仿真时间、步长等
fmi3EnterInitializationMode ()	进入初始化模式
fmi3ExitInitializationMode ()	退出初始化模式
fmi3Terminate ()	仿真出现错误，终止
fmi3FreeInstance ()	释放 GISSMU 模型实例
fmi3Reset ()	恢复至实例化阶段
fmi3SetXXX ()	设置参数，XXX 代表 Real、Integer、Boolean、String
fmi3GetXXX ()	获取参数，XXX 代表 Real、Integer、Boolean、String
fmi3DoStep ()	运行当前 GISSMU 模型，执行一个步长并获取结果
fmi3CancelStep ()	取消当前步长的运行
fmi3GetStatus ()	获取状态，包括 Real、Integer、Boolean、String 所有类型的状态
fmi3GetXXXStatus ()	获取状态，XXX 代表 Real、Integer、Boolean、String
fmiLastSuccessfulTime ()	上一次仿真成功时间

A. 9 接口属性

表A. 6 接口属性

接口属性
ModelIdentifier:normalizedString
needExecutionTool:boolean=false
canBeInstantiatedOnlyOncePerProcess:boolean=false
canGetAndSetGISSMUState:boolean=false
canSerializeGISSMUState:boolean=false
providesDirectionalDerivatives:boolean=false
providesAdjointDerivatives:boolean=false
canHandleVariableCommunicationStepSize:boolean=false
fixedInternalStepSize:double
maxOutputDerivativeOrder:unsignedInt=0
recommendIntermediateInputSmoothness:int=0
providesIntermediateUpdate:boolean=false
mightReturnEarlyFromDoStep:boolean=false
canReturnEarlyAfterIntermediateUpdate:boolean=false
hasEventMode:boolean=false
providesEvaluatedDiscreteStates:boolean=false

附录 B (资料性) 船舶动力定位系统示例

下面是一个具体的船舶方向示例，包括生成GISSMU和编写modelDescription.xml文件的详细步骤。我们将以一个简单的船舶动力定位系统为例，该系统包含风传感器模型、GNSS传感器模型和控制模型。

B.1 船舶动力定位系统概述

B.1.1 风传感模型 (Wind Sensor Model)

输入：风速和风向

输出：NMEA MWV消息

B.1.2 GNSS传感器模型 (GNSS Sensor Model)

输入：船舶位置

输出：NMEA GGA和GST消息

B.1.3 控制模型 (Control Model)

输入：NMEA MWV、GGA和GST消息

输出：舵机和推进器指令

B.2 生成 GISSMU

B.2.1 生成风传感器模型的GISSMU

使用支持FMI标准的建模工具（如Dymola、Modelica等）创建模型。
导出模型为GISSMU格式。

B.2.2 生成GNSS传感器模型的GISSMU

同上，使用建模工具创建模型并导出为GISSMU格式。

B.2.3 生成控制模型的GISSMU

同上，使用建模工具创建模型并导出为GISSMU格式。
编写modelDescription.xml

B.2.3.1 风传感器模型的 modelDescription.xml

```
<? xml version = "1.0" encoding = "utf-8" ? >

<modelDescription xmlns = "http://schemas.ieee.org/GISSMU/2.0/CoSimulation" fmiVersion = "2.0" modelName =
"WindSensorModel" guid = "{a1b2c3d4-e5f6-7890-1234-567890abcdef}">

  <CoSimulation modelIdentifier = "WindSensorModel">

    <SourceFiles>

      <File name = "WindSensorModel.dll" />

    </ SourceFiles>

  </ CoSimulation>

  <ModelVariables>

    <ScalarVariable name = "wind_speed" valueReference = "0" causality = "input" variability = "continuous" initial =
```

```

"exact">
    <Real start = "0.0" />
</ ScalarVariable>

<ScalarVariable name = "wind_direction" valueReference = "1" causality = "input" variability = "continuous" initial
= "exact">
    <Real start = "0.0" />
</ ScalarVariable>

<ScalarVariable name = "nmea_mwv" valueReference = "2" causality = "output" variability = "discrete">
    <String />
</ ScalarVariable>
</ ModelVariables>
</ modelDescription>

```

B. 2. 3. 2 GNSS 传感器模型的 modelDescription.xml

```

<? xml version = "1.0" encoding = "utf-8" ?>
<modelDescription xmlns = "http://schemas.ieee.org/GISSMU/2.0/CoSimulation" fmiVersion = "2.0" modelName =
"GnsSensorModel" guid = "{b2c3d4e5-f678-90ab-cdef-1234567890ab}">
    <CoSimulation modelIdentifier = "GnsSensorModel">
        <SourceFiles>
            <File name = "GnsSensorModel.dll" />
        </ SourceFiles>
    </ CoSimulation>
    <ModelVariables>
        <ScalarVariable name = "latitude" valueReference = "0" causality = "input" variability = "continuous" initial =
"exact">
            <Real start = "0.0" />
        </ ScalarVariable>

        <ScalarVariable name = "longitude" valueReference = "1" causality = "input" variability = "continuous" initial =
"exact">
            <Real start = "0.0" />
        </ ScalarVariable>

        <ScalarVariable name = "nmea_gga" valueReference = "2" causality = "output" variability = "discrete">
            <String />
        </ ScalarVariable>
    </ ModelVariables>
</ modelDescription>

```

```

<ScalarVariable name = "nmea_gst" valueReference = "3" causality = "output" variability = "discrete">
  <String />
</ScalarVariable>
</ModelVariables>
</modelDescription>
B. 2. 3. 3 控制模型的 modelDescription.xml
<? xml version = "1.0" encoding = "utf-8" ?>
<modelDescription xmlns = "http://schemas.ieee.org/GISSMU/2.0/CoSimulation" fmiVersion = "2.0" modelName =
"ControlModel" guid = "{c3d4e5f6-g7h8-90ij-k1l2-mnopqrstuvwx}">
  <CoSimulation modelIdentifier = "ControlModel">
    <SourceFiles>
      <File name = "ControlModel.dll" />
    </SourceFiles>
  </CoSimulation>
  <ModelVariables>
    <ScalarVariable name = "nmea_mwv" valueReference = "0" causality = "input" variability = "discrete" initial =
"exact">
      <String />
    </ScalarVariable>
    <ScalarVariable name = "nmea_gga" valueReference = "1" causality = "input" variability = "discrete" initial =
"exact">
      <String />
    </ScalarVariable>
    <ScalarVariable name = "nmea_gst" valueReference = "2" causality = "input" variability = "discrete" initial =
"exact">
      <String />
    </ScalarVariable>
    <ScalarVariable name = "rudder_command" valueReference = "3" causality = "output" variability = "continuous">
      <Real />
    </ScalarVariable>
    <ScalarVariable name = "thruster_command" valueReference = "4" causality = "output" variability = "continuous">
      <Real />
    </ScalarVariable>
  </ModelVariables>
</modelDescription>

```

```
</ModelVariables>
</modelDescription>
```

B.2.3.4 编写 OspModelDescription.xml

接下来，需要定义这些模型的接口，并将它们组合成一个系统结构。

风传感器模型的 OspModelDescription.xml

```
<? xml version = "1.0" encoding = "utf-8" ? >
<OspModelDescription xmlns = "https://open-simulation-platform.com/OspModelDescription/1.0.0" version = "1.0">
  <VariableGroups>
    <Generic name = "wind_sensor_inputs">
      <Variable ref = "wind_speed" />
      <Variable ref = "wind_direction" />
    </Generic>
    <Generic name = "nmea_mwv_output">
      <Variable ref = "nmea_mwv" />
    </Generic>
  </VariableGroups>
</OspModelDescription>
```

B.2.3.5 GNSS 传感器模型的 OspModelDescription.xml

```
<? xml version = "1.0" encoding = "utf-8" ? >
<OspModelDescription xmlns = "https://open-simulation-platform.com/OspModelDescription/1.0.0" version = "1.0">
  <VariableGroups>
    <Generic name = "gnss_sensor_inputs">
      <Variable ref = "latitude" />
      <Variable ref = "longitude" />
    </Generic>
    <Generic name = "nmea_gga_output">
      <Variable ref = "nmea_gga" />
    </Generic>
    <Generic name = "nmea_gst_output">
      <Variable ref = "nmea_gst" />
    </Generic>
  </VariableGroups>
</OspModelDescription>
```

B.2.3.6 控制模型的 OspModelDescription.xml

```

<? xml version = "1.0" encoding = "utf-8" ? >

<OspModelDescription xmlns = "https://open-simulation-platform.com/OspModelDescription/1.0.0" version = "1.0">

  <VariableGroups>

    <Generic name = "control_inputs">

      <Variable ref = "nmea_mwv" />

      <Variable ref = "nmea_gga" />

      <Variable ref = "nmea_gst" />

    </ Generic>

    <Generic name = "control_outputs">

      <Variable ref = "rudder_command" />

      <Variable ref = "thruster_command" />

    </ Generic>

  </ VariableGroups>

</ OspModelDescription>

```

B.2.3.7 编写 OspSystemStructure.xml

最后，需要定义系统结构，将各个模型连接起来。

```

<? xml version = "1.0" encoding = "utf-8" ? >

<OspSystemStructure xmlns = "https://open-simulation-platform.com/OspModelDescription/1.0.0" version = "1.0">

  <Models>

    <Model name = "wind_sensor_model" path = "path/to/wind_sensor_model.GISSMU" />

    <Model name = "gnss_sensor_model" path = "path/to/gnss_sensor_model.GISSMU" />

    <Model name = "control_model" path = "path/to/control_model.GISSMU" />

  </ Models>

  <Connections>

    <Connection from = "wind_sensor_model.nmea_mwv_output" to = "control_model.control_inputs" />

    <Connection from = "gnss_sensor_model.nmea_gga_output" to = "control_model.control_inputs" />

    <Connection from = "gnss_sensor_model.nmea_gst_output" to = "control_model.control_inputs" />

  </ Connections>

</ OspSystemStructure>

```

通过以上步骤，可以生成并验证船舶动力定位系统的GISSMU模型，并将它们组合成一个完整的系统进行仿真。

附录 C (资料性) 验证示例代码

C.1 填充本体示例

```

<!--OspModelDescription.xml-->
<ModelDescription>
  <SimulationInformation>
    <ModelVariables>
      <ScalarVariable name = "force" valueReference = "1" causality = "input" variability = "continuous" initial =
"exact">
        <Real />
      </ScalarVariable>
      <ScalarVariable name = "velocity" valueReference = "2" causality = "output" variability = "continuous" initial =
"approx">
        <Real />
      </ScalarVariable>
    </ModelVariables>
    <ModelStructure>
      <VariableGroups>
        <VariableGroup name = "linearMechanicalPort">
          <Variable name = "force" valueReference = "1" />
          <Variable name = "velocity" valueReference = "2" />
        </VariableGroup>
      </VariableGroups>
    </ModelStructure>
  </SimulationInformation>
</ModelDescription>

<!--OspSystemStructure.xml-->
<SystemStructure>
  <Simulators>
    <Simulator name = "sim1" source = "path/to/sim1.GISSMU" />
    <Simulator name = "sim2" source = "path/to/sim2.GISSMU" />
  </Simulators>
</SystemStructure>

```

```

</ Simulators>

<Connections>

    <VariableGroupConnection name = "conn1" simulator1 = "sim1" variableGroup1 = "linearMechanicalPort"
simulator2 = "sim2" variableGroup2 = "linearMechanicalPort" />

</ Connections>
</ SystemStructure>

```

填充本体示例代码:

// 创建本体工厂和本体管理器

```
OWLOntologyManager manager = OWLManager.createOWLOntologyManager();
```

```
OWLDataFactory factory = manager.getOWLDataFactory();
```

// 加载 OSP-IS 本体

```
IRI ontologyIRI = IRI.create("https://data.dnvgl.com/osp/osp.owl");
```

```
OWLOntology ontology = manager.loadOntologyFromOntologyDocument(ontologyIRI);
```

// 创建模拟器个体

```
OWLNamedIndividual sim1 = factory.getOWLNamedIndividual(IRI.create("http://example.org/sim1"));
```

```
OWLNamedIndividual sim2 = factory.getOWLNamedIndividual(IRI.create("http://example.org/sim2"));
```

// 创建变量组个体

```
OWLNamedIndividual linearMechanicalPort1 =
```

```
factory.getOWLNamedIndividual(IRI.create("http://example.org/linearMechanicalPort1"));
```

```
OWLNamedIndividual linearMechanicalPort2 =
```

```
factory.getOWLNamedIndividual(IRI.create("http://example.org/linearMechanicalPort2"));
```

// 创建变量个体

```
OWLNamedIndividual force1 = factory.getOWLNamedIndividual(IRI.create("http://example.org/force1"));
```

```
OWLNamedIndividual velocity1 = factory.getOWLNamedIndividual(IRI.create("http://example.org/velocity1"));
```

```
OWLNamedIndividual force2 = factory.getOWLNamedIndividual(IRI.create("http://example.org/force2"));
```

```
OWLNamedIndividual velocity2 = factory.getOWLNamedIndividual(IRI.create("http://example.org/velocity2"));
```

// 设置变量属性

```
OWLClass variableClass = factory.getOWLClass(IRI.create("http://example.org/Variable"));
```

```
OWLObjectProperty hasName = factory.getOWLObjectProperty(IRI.create("http://example.org/hasName"));
```

```

OWLDataProperty hasDataType = factory.getOWLDataProperty(IRI.create("http://example.org/hasDataType"));
OWLDataProperty hasUnit = factory.getOWLDataProperty(IRI.create("http://example.org/hasUnit"));
OWLDataProperty hasCausality = factory.getOWLDataProperty(IRI.create("http://example.org/hasCausality"));

```

```

manager.addAxiom(ontology, factory.getOWLObjectPropertyAssertionAxiom(variableClass, force1));
manager.addAxiom(ontology, factory.getOWLObjectPropertyAssertionAxiom(hasName, force1,
factory.getOWLLiteral("force")));
manager.addAxiom(ontology, factory.getOWLObjectPropertyAssertionAxiom(hasDataType, force1,
factory.getOWLLiteral("Real")));
manager.addAxiom(ontology, factory.getOWLObjectPropertyAssertionAxiom(hasUnit, force1,
factory.getOWLLiteral("N")));
manager.addAxiom(ontology, factory.getOWLObjectPropertyAssertionAxiom(hasCausality, force1,
factory.getOWLLiteral("input")));

```

// 创建连接关系

```

OWLClass variableGroupConnectionClass =
factory.getOWLClass(IRI.create("http://example.org/VariableGroupConnection"));
OWLObjectProperty hasSimulator1 = factory.getOWLObjectProperty(IRI.create("http://example.org/hasSimulator1"));
OWLObjectProperty hasVariableGroup1 =
factory.getOWLObjectProperty(IRI.create("http://example.org/hasVariableGroup1"));
OWLObjectProperty hasSimulator2 = factory.getOWLObjectProperty(IRI.create("http://example.org/hasSimulator2"));
OWLObjectProperty hasVariableGroup2 =
factory.getOWLObjectProperty(IRI.create("http://example.org/hasVariableGroup2"));

```

```

OWLNamedIndividual conn1 = factory.getOWLNamedIndividual(IRI.create("http://example.org/conn1"));
manager.addAxiom(ontology, factory.getOWLObjectPropertyAssertionAxiom(variableGroupConnectionClass, conn1));
manager.addAxiom(ontology, factory.getOWLObjectPropertyAssertionAxiom(hasSimulator1, conn1, sim1));
manager.addAxiom(ontology, factory.getOWLObjectPropertyAssertionAxiom(hasVariableGroup1, conn1,
linearMechanicalPort1));
manager.addAxiom(ontology, factory.getOWLObjectPropertyAssertionAxiom(hasSimulator2, conn1, sim2));
manager.addAxiom(ontology, factory.getOWLObjectPropertyAssertionAxiom(hasVariableGroup2, conn1,
linearMechanicalPort2));

```

// 保存本体

```
IRI saveIRI = IRI.create("file:/path/to/osp-populated.owl");
manager.saveOntology(ontology, saveIRI);
```

G.2 分类验证示例代码

// 加载填充好的本体

```
IRI populatedOntologyIRI = IRI.create("file:/path/to/osp-populated.owl");
OWLOntology populatedOntology = manager.loadOntologyFromOntologyDocument(populatedOntologyIRI);
```

// 创建 Reasoner

```
OWLReasonerFactory reasonerFactory = new HermiT.ReasonerFactory();
OWLReasoner reasoner = reasonerFactory.createReasoner(populatedOntology);
```

// 获取错误类

```
OWLClass invalidVariableConnectionClass =
factory.getOWLClass(IRI.create("http://example.org/InvalidVariableConnection"));
Set<OWLNamedIndividual> invalidVariableConnections = reasoner.getInstances(invalidVariableConnectionClass,
false).getFlattened();
```

// 生成错误消息

```
for (OWLNamedIndividual invalidConnection : invalidVariableConnections) {
    String message = "Invalid Variable Connection: " + invalidConnection.getIRI().toString();
    System.out.println(message);
}
```

// 关闭 Reasoner

```
reasoner.dispose();
```