ADD第一次迭代

Step 1: Confirm there is sufficient requirements information

* 1. Functional Requirements

1. 系统能够验证客户请求的服务的可用性
2. 系统需要有效地管理好时间和日期信息，因为客户可能会请求服务的变更信息
   1. Design Constraints
3. 在代理工作基站不能设有持久化数据的缓存，以限制本地故障造成的影响
4. 在办公室的位点不能有数据库
   1. 本地办公室不设有管理员
   2. 本地办公室没有停机维修时间
5. 采用具备优良工程化操作的体系结构
6. 不能通过高容错的硬件来实现高可用性
   1. Quality Attribute Requirements
7. 系统能够支持一个会话中同时提供多个服务

|  |  |
| --- | --- |
| Element | Statement |
| Stimulus | 客户在一次通话过程中请求了多个服务，某个服务中途发生了失效 |
| Stimulus source | 服务部件 |
| Environment | 系统在与客户进行交互的过程中，特别是客户在使用发生失效服务的过程中 |
| Artifact | C4系统 |
| Response | 系统迅速检测到服务失效的情况，并且提示客户等待修复服务，然后迅速恢复失效的服务 |
| Response measure | 系统检测到服务失效的时间距离服务失效发生的时间不超过1秒；  系统恢复服务的时间距离提示用户服务失效的时间不超过1秒；  系统从检测到服务失效到为用户恢复失效服务的时间不超过2秒 |

1. 系统能够支持会话中断和长延时会话

|  |  |
| --- | --- |
| Element | Statement |
| Stimulus | 客户在和系统通话的过程中请求中断或直接中断通话；  客户在和系统进行长延时通话过程中连接意外中断； |
| Stimulus source | 客户或连接部件的失效 |
| Environment | 系统在与客户通话的过程中 |
| Artifact | C4系统 |
| Response | 系统保存客户本次会话至中断处的请求服务信息，如果通话由客户主动中断则保留中断前的通话信息一段时间，直到客户重新连接请求通话时恢复上次通话内容；  系统在通话被动中断的情况下会向客户重新发送通话请求，如果客户拒绝或长时间魏回应则按照客户主动中断通话的情况处理，否则将通话信息恢复到中断时的状态 |
| Response measure | 系统需要在0.5秒内检测到通话断开，以便及时进行后续处理，保证用户通话的流畅性；  系统需要确保90%的通话信息被保存下来，以保证与客户通话的完整性 |

1. 系统需要以小客户群为基础，采取经济上可行的架构，且考虑该小客户群未来会发 展为巨大客户网络的情况

|  |  |
| --- | --- |
| Element | Statement |
| Stimulus | 系统需要扩充客户容量，提高并发性 |
| Stimulus source | 客户量增加；  系统硬件的更新；  系统软件的编码优化 |
| Environment | 任何时刻 |
| Artifact | C4系统 |
| Response | 系统更换运行的硬件平台；  系统重新编译运行 |
| Response measure | 系统的软硬件更新时长要限制在4小时/2个月 |

1. 能进行身份识别、监视

|  |  |
| --- | --- |
| Element | Statement |
| Stimulus | 客户在和系统进行交互，期间向系统请求服务或更改配置 |
| Stimulus source | 客户 |
| Environment | 系统与客户交互过程中 |
| Artifact | C4系统 |
| Response | 系统需要对请求服务的客户身份进行识别验证，防止匿名请求服务；  系统在和客户协同完成更改配置等任务时需要监控客户的配置数据，防止恶意修改配置 |
| Response measure | 系统要在0.5秒内检测到客户的恶意行为并发出警告；  系统在检测到客户恶意行为后要将客户账户冻结1小时，并记录相应客户信息，在后续若该客户仍有恶意行为则冻结时间呈指数增加；  系统对客户身份信息的识别要在2秒内完成，若检测不到则拒绝提供请求的服务 |

1. 未来需要消除瓶颈

|  |  |
| --- | --- |
| Element | Statement |
| Stimulus | 系统维护人员要优化系统程序中的瓶颈部件 |
| Stimulus source | 系统维护人员 |
| Environment | 系统维护时 |
| Artifact | C4系统 |
| Response | 系统的工程结构能够允许维护人员对系统进行修改优化 |
| Response measure | 系统的瓶颈消除优化需要在4小时/2个月的时间中完成 |

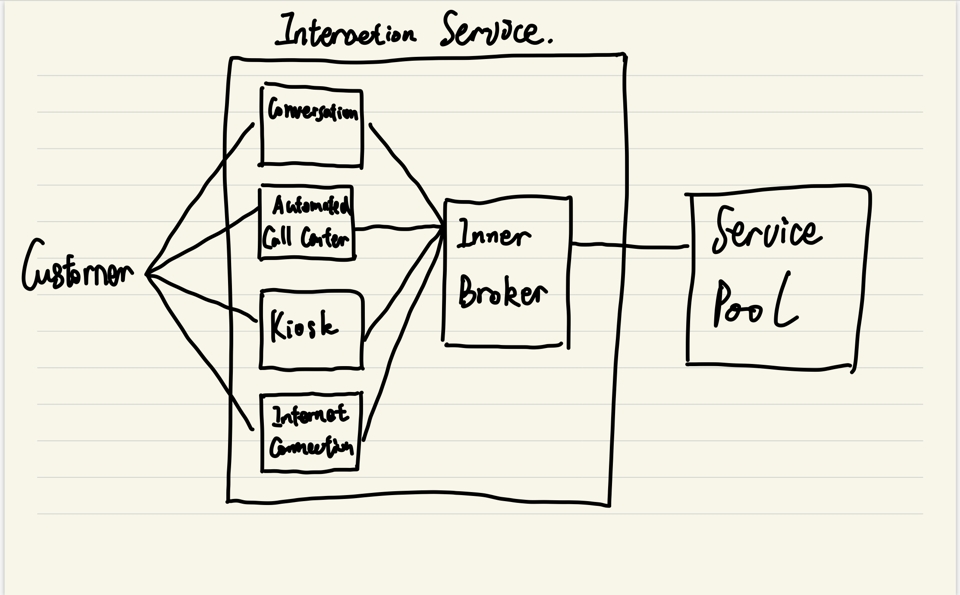
1. 能够同时支持大量代表的服务请求（至少100个）

|  |  |
| --- | --- |
| Element | Statement |
| Stimulus | 系统运行时某一时刻或某一时段同时有多个客户请求服务 |
| Stimulus source | 客户 |
| Environment | 系统与多个客户交互过程中，客户同时请求服务时 |
| Artifact | C4系统 |
| Response | 系统验证客户请求服务的可用性后能够为所有客户提供服务 |
| Response measure | 系统至少能同时支持100个客户使用同一个服务资源；  系统对于请求服务的响应时间平均不超过5秒； |

1. 几乎7x24小时的应用可用性

|  |  |
| --- | --- |
| Element | Statement |
| Stimulus | 客户在某一时刻请求连接发起会话 |
| Stimulus source | 客户 |
| Environment | 系统全时（任何时候） |
| Artifact | C4系统 |
| Response | 系统若可用，则允许客户连接，建立通话，否则提示客户等待或取消连接 |
| Response measure | 系统必须在99.9%的时间中可用，即一周最多10分钟宕机，客户最长等待时间不超过0.5秒（最坏情况下一周有1200个用户等待了系统恢复） |

Step 2: Choose an element of the system to decompose



第一次迭代选取C4系统的Interaction Service来进行分解。

Step 3: Identify candidate architectural drivers

下表给出了10个架构驱动的元素。其中有7个是从整体系统的非功能性需求中识别出来的，有2个是从与该子系统相关的功能性需求中识别出来的，1个是从整体系统的设计限制中识别出来的。

·保留下来的架构驱动元素都是对Interaction Service模块的影响达到Medium或High的，他们都与直接和客户交互通话的过程有关，直接影响了客户与系统交互的体验，也对系统内部件的选择有比较大的影响

·删去的架构驱动元素#3、#5主要和系统的可修改性有关，而该子系统本身已经将功能进行内聚，理论上具有较良好的可修改性；#8、#9则是对整体模块的元素没有太大的影响，具体的实现仅仅需要额外增加一个部件，而不必太多的依赖于其他部件，同时其他部件也对其依赖很少，因此对于Interaction Service子系统的影响不大

|  |  |  |  |
| --- | --- | --- | --- |
| # | Architectural  Drivers | Importance | Difficulty |
| 1 | Scenario 1  多服务支持 | High | High |
| 2 | Scenario 2  延时与中断支持 | High | Medium |
| ~~3~~ | ~~Scenario 3~~  ~~客户容量增加~~ | ~~Low~~ | ~~Medium~~ |
| 4 | Scenario 4  身份识别与监视 | High | High |
| ~~5~~ | ~~Scenario 5~~  ~~消除瓶颈~~ | ~~Low~~ | ~~Medium~~ |
| 6 | Scenario 6  支持多并行客户代表 | High | High |
| 7 | Scenario 7  7x24小时可用性 | High | Medium |
| ~~8~~ | ~~Requirement 1~~  ~~验证服务可用性~~ | ~~Low~~ | ~~Low~~ |
| ~~9~~ | ~~Requirement 2~~  ~~提供更新版本信息~~ | ~~Low~~ | ~~Low~~ |
| 10 | Design Constraint 1  不能使用高容错硬件 | Medium | High |

Step 4: Choose a design concept that satisfies the architectural drivers

·Substep 1: Identify Design Concerns

|  |  |  |
| --- | --- | --- |
| Quality Attribute | Design Concern | Subroutine Concern |
| Availability | 提供多服务保障 | 检测服务可用性 |
| 恢复服务 |
| 提供中断和长延时通话支持 | 检测通话连接状态 |
| 保存通话信息 |
| 几乎7x24小时可用 | 检测系统失效 |
| 失效防备与修复 |
| Scalability | 同时支持多客户服务请求 | 保证系统的承载能力 |
| Security | 身份验证和行为监控 | 客户请求服务时的身份验证 |
| 反映行为的数据监控分析 |

·Substep 2 & Substep 3: List Alternative Patterns for Subordinate Concerns & Select Patterns from the List

1. 提供多服务保障
   1. 检测服务可用性

|  |  |  |  |
| --- | --- | --- | --- |
| # | 模式 | 信息量 | 响应时间 |
| 1 | Ping/Echo | 1 | 0.8 S |
| 2 | Heartbeat | 2 | 0.5 S |
| 3 | Self-Test | 3 | 0.1 S |

选择模式为Heartbeat：Ping/Echo的0.8s虽然在1s的限定范围内，但是Ping/Echo并不是周期性的检测，不能保证每次检测都是Ping时系统有效，但收到Echo立刻检测到系统失效的情况，而且如果Echo发出后系统立刻失效，则需要等到下一次Ping才能检测出系统失效的情况，不确定性较大；Self-Test虽然拥有最快的响应时间，但他需要的信号量是最多的，而且自检放在被检测系统中会增加系统本身的负担，影响系统其他功能的表现性能；综合来看Heartbeat具备周期性检测，信号量产生的负载适中，同时响应时间符合1s的要求。

* 1. 恢复服务

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | 模式 | 服务信息恢复度 | 恢复时间 | 系统负载度 |
| 1 | Active Redundancy | 100% | 0.2 S | 3 |
| 2 | Passive Redundancy | 90% | 0.7 S | 2 |
| 3 | Spare | 50% | 1 S | 0 |

选择模式为Passive Redundancy：Active Redundancy虽然具备最好的信息恢复度和最快的恢复时间，但是对系统产生的负载量也由于运行时的重复输入执行而更高，会影响到系统的实时性；Spare虽然具备擦边球式的恢复时间，对系统产生的负载也由于其外部备用部件特性几乎为零，但对于信息恢复的能力仅有50%甚至不确定，这会对用户的通话完整性体验造成较大困扰；综合来看选择恢复时间合格，系统负载相对适中，同时能够尽可能保证客户通话完整性的Passive Redundancy模式。

1. 提供中断和长延时通话支持
   1. 检测通话连接状态

|  |  |  |  |
| --- | --- | --- | --- |
| # | 模式 | 信息量 | 响应时间 |
| 1 | Ping/Echo | 1 | 0.8 S |
| 2 | Heartbeat | 2 | 0.5 S |
| 3 | Self-Test | 3 | 0.1 S |

选择模式为Self-Test模式：显然Ping/Echo的响应时间不能满足0.5s的阈值要求，而Heartbeat虽然刚好为0.5s的响应时间，但心跳之间的时间间隙会让实际响应时间≥0.5s；综合来看，虽然Self-Test信号量造成的负载较大，但该关注点对系统的重要性较高，有其硬实时要求，也算是好钢用在刀刃上的选择。

* 1. 保存通话信息

|  |  |  |  |
| --- | --- | --- | --- |
| # | 模式 | 通话信息恢复度 | 系统负载 |
| 1 | Active Redundancy | 100% | 3 |
| 2 | Passive Redundancy | 90% | 2 |
| 3 | Checkpoint | 90% | 1 |

选择模式为Checkpoint模式：不选择Active Redundancy模式的原因依然是其对系统增加的负载过高，影响了系统整体的性能；此处不选择Passive Redundancy的原因是因为Checkpoint比起它更加灵活，而且Checkpoint带来的系统负载也比它要低；简单来说，Checkpoint能够提供的信息恢复度具有浮动性，因为能够自定义checkpoint的频率，从而控制每一次checkpoint写入的信息量有多少，也就是如果checkpoint频率足够高，它甚至能具有Active Redundancy的信息恢复能力（但这样checkpoint附带的计时器等部件综合开销会高于Active Redundancy）；总而言之，checkpoint兼具较好的灵活性和系统低负载性，在对于部件的恢复要求不高而对数据恢复要求高的情况下是比较好的选择。

1. 几乎7x24小时可用
   1. 检测系统失效

|  |  |  |  |
| --- | --- | --- | --- |
| # | 模式 | 信息量 | 响应时间 |
| 1 | Ping/Echo | 1 | 0.8 S |
| 2 | Heartbeat | 2 | 0.5 S |
| 3 | Self-Test | 3 | 0.1 S |
| 4 | Exception Detection | 4 | 1 S |

选择模式为Exception Detection：正如Exception Detection所描述 - 有时候发现错误最好的方法是在这个错误发生后再检测到它。前三者虽然响应时间非常快速，但是在7x24小时的跨度内会对系统产生灾难性的负载量，从而严重影响到系统的整体性能。而Exception Detection在仅当用户连接失败时检测到系统失效相比而言开销更小，且效果相差无几；而且采用这种模式还能够更加灵活地设计检错机制，例如可以设立一个伪用户端，按照一定的频率尝试连接系统以模拟用户的连接行为进而达到检错目的。

* 1. 失效防备与修复

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | 模式 | 系统配置恢复度 | 恢复时间 | 系统负载度 |
| 1 | Active Redundancy | 100% | 0.2 S | 3 |
| 2 | Passive Redundancy | 90% | 0.7 S | 2 |
| 3 | Spare | 50% | 1 S | 0 |
| 4 | Reconfiguration | 0% | 0.4 S | 1 |

选择模式为Active Redundancy：Passive Redundancy和Spare均不能满足0.5s的硬实时要求；Reconfiguration虽然恢复时间小于0.5s且给系统带来的负载也比较低，但是其对于系统配置数据的恢复程度为0%，这可能导致系统保留的某些用户模型被清空，给系统易用性等带来不好的影响，对用户而言也相当于从头开始使用系统，无异于可用性的降低。

1. 同时支持多客户服务请求
   1. 保证系统的承载能力

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | 模式 | 开发难度 | 实时性能 | 多次迭代性 |
| 1 | Trade-off with  Performance | 低 | 差 | 极低 |
| 2 | Avoid Single Point Failure | 高 | 优 | 中 |
| 3 | Lightweight Core | 较高 | 优 | 低 |
| 4 | Asynchronous Interaction | 中 | 中 | 较高 |

选择模式为Asynchronous Interaction：Trade-off with Performance的问题在于它虽然简单，但治标不治本，它以降低系统性能的方式来降低系统的负担，那么随着用户的增加系统必然面临第二次优化，显然不停地降低Performance不是好的选择；Avoid Single Point Failure的问题在于对每个部件的面面俱到的考虑方式和设计模式会让系统的复杂度随着每一次优化迭代而提高，从而对于不断需要迭代优化的系统而言不是好的选择，而且它本身的开发难度也比较高；Lightweight Core的问题则是轻量级核心组件带来的是大量周围部件的协同，进而提高系统的复杂度，也会让系统的可修改性逐渐降低；总而言之，在对实时性能的要求不是很高（5s），同时未来存在多次迭代的潜在需求时，Asynchronous Interaction就是比较好的选择（此处指的是和用户交互上的Asynchronous）。

1. 身份验证和行为监控
   1. 客户请求服务时的身份验证

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | 模式 | 安全程度 | 成本 | 耗时 |
| 1 | Identify Actors | 低 | 低 | 0.5 S |
| 2 | Authenticate Actors | 中 | 中 | 1 S |
| 3 | Authorize Actors | 高 | 高 | 1 S + 0.8 S |

选择模式为Authorize Actors：Identify Actors仅仅是对接入的客户身份进行了识别记录，虽然在客户做出恶意行为后能够加以阻挡再次访问，但无法抵御其第一次恶意访问；Authenticate Actors虽然安全度稍有提高，成本也比较适中，但是无法防备客户的账号被盗用的情况，该情况一旦发生对客户和服务端的破坏都比较严重；选择Authorize Actors则是在客户经过身份验证后再在客户请求服务时进行了一次动态授权，安全性更为严密，尽管成本相对偏高，但对于系统和客户都能提供较为完备的保护。

* 1. 反映行为的数据监控分析

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | 模式 | 检测时间 | 成本 | 准确度 |
| 1 | Verify Message Integrity | 0.3 S | 高 | 90% |
| 2 | Detect Message Delay | 0.1 S | 低 | 60% |
| 3 | Maintain Audit Trail | 用户交互结束 | 偏高 | 100% |

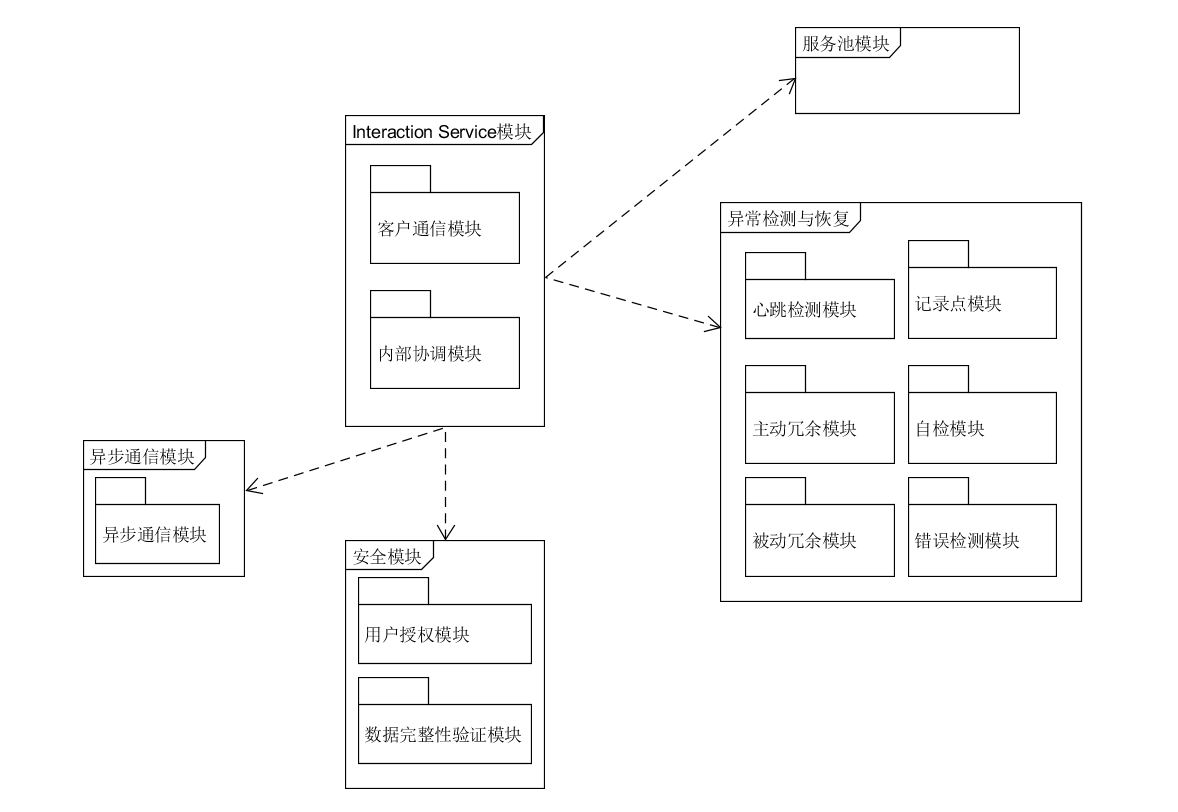
选择模式为Verify Message Integrity：首先排除Maintain Audit Trail，因为它的检测要等到用户交互结束后，检查本次记录才能分析出用户的恶意行为，明显不符合要求的0.5秒阈值；Detect Message Delay虽然检测时间短，成本也低，但是准确率会由于如用户网络连接差等不可抗力原因造成的误判而下降，从而降低用户体验；而Verify Message Integrity的检测时间在要求范围内，且成本只在于计算message的hash值等，其“高”是相对于另外两个备选方案而言，同时又有相对较高的准确度。

·Substep 4: Determine Relationship Between Patterns and Drivers

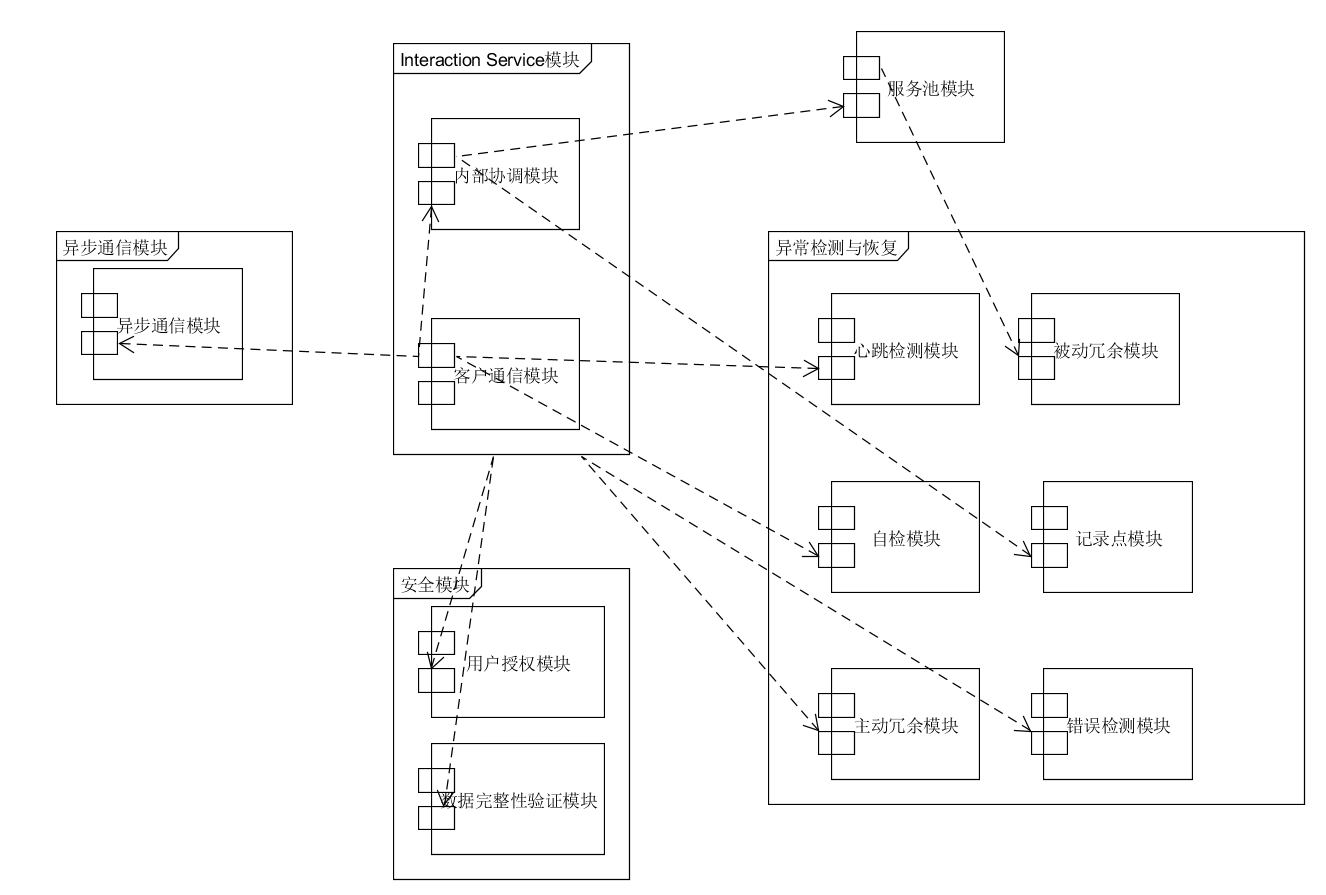
|  |  |  |  |
| --- | --- | --- | --- |
| # | Subroutine Concern | Pattern | Attribute Driven |
| 1 | 检测服务可用性 | Heartbeat | Scenario 1：提供多服务支持 |
| 2 | 恢复服务 | Passive Redundancy | Scenario 1：提供多服务支持 |
| 3 | 检测通话连接状态 | Self-Test | Scenario 2：提供中断和长延时通话支持 |
| 4 | 保存通话信息 | Checkpoint | Scenario 2：提供中断和长延时通话支持 |
| 5 | 检测系统失效 | Exception Detection | Scenario 7：几乎7x24小时的应用可用性 |
| 6 | 失效防备与修复 | Active Redundancy | Scenario 7：几乎7x24小时的应用可用性 |
| 7 | 保证系统的承载能力 | Asynchronous Interaction | Scenario 6：同时支持多客户服务请求 |
| 8 | 客户请求服务时的身份验证 | Authorize Actors | Scenario 4：身份验证和行为监控 |
| 9 | 反映行为的数据监控分析 | Verify Message Integrity | Scenario 4：身份验证和行为监控 |

·Substep 5：Capture Preliminary Architectural Views

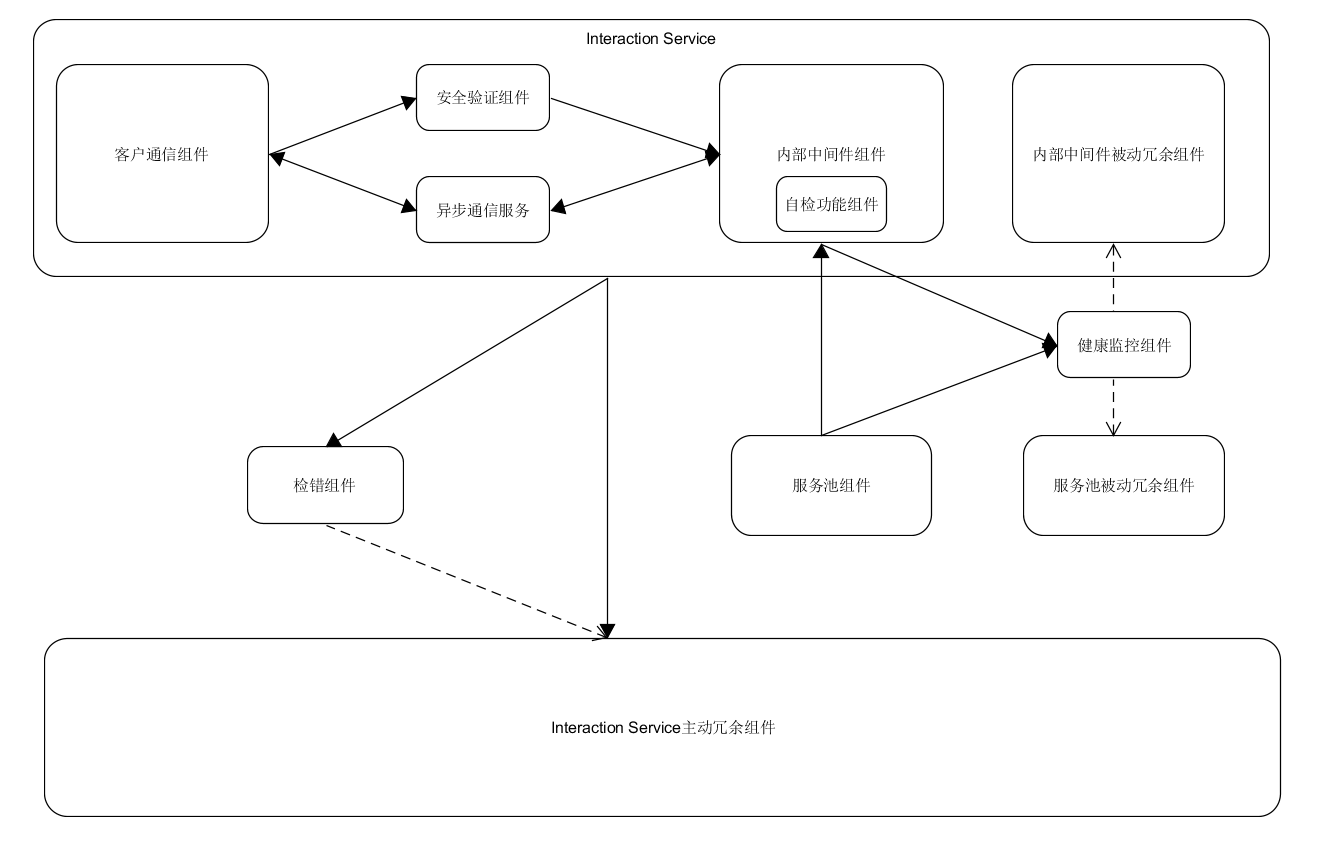
Module View



C&C View



Software Element View



·Substep 6：Evaluate and Resolve Inconsistencies

上述过程中没有违反Constraint 1提到的“不能使用fault-tolerant硬件”的约束，其他的设计之间也没有发生冲突，并且已经覆盖了关于该element的全部架构驱动因素。

Step 5: Instantiate architectural elements and allocate responsibilities

·5.1 Interaction Service：负责和客户进行通话交互，负责代理客户向服务池请求服务

·5.2 服务池：负责提供相关服务

·5.3 异常与恢复：负责对系统、系统模块进行检测，负责在系统出现异常时进行修复

·5.4 异步通信：负责提供客户与系统之间的异步通信渠道

·5.5 安全：负责检测客户的恶意行为，负责验证客户数据的完整性

Step 6: Define interfaces for instantiated elements

|  |  |  |  |
| --- | --- | --- | --- |
| # | 模块 | 接口 | 职责 |
| 1 | Interaction Service | Socket startConversation(User user)  Service requestService(Integer id) | 5.1 |
| 2 | 服务池 | Service getService(Integer id) | 5.2 |
| 3 | 异常与恢复 | Void receiveHeartbeat(Boolean signal)  Void selfTest(Status status)  Boolean hasException(Status status)  Void checkpoint(Boolean signal)  Void redundancy() | 5.3 |
| 4 | 异步通信 | Socket asyncCommunication(User user) | 5.4 |
| 5 | 安全 | Boolean isAuthorized(User user)  Boolean isValid(Message msg) | 5.5 |

Step 7: Verify and refine requirements and make them constraints for instantiated elements

经过对视图、职责表、职责描述和与架构驱动表和设计关注点的比对与审查，确认本次迭代的设计没有遗漏。